Techniques to Secure HW/SW-Programmable SoC Architectures for Edge Computing

Sicherheitstechniken für die Datenverarbeitung an der Edge auf HW/SW-programmierbaren SoC-Architekturen

Der Technischen Fakultät
der Friedrich-Alexander-Universität
Erlangen-Nürnberg
zur
Erlangung des Doktorgrades Dr.-Ing.

vorgelegt von

Franz-Josef Streit
Als Dissertation genehmigt
von der Technischen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg
Tag der mündlichen Prüfung: 25. II. 2022

Gutachter: Prof. Dr.-Ing. Jürgen Teich
Prof. Dr.-Ing. Georg Sigl
"Das – ist mein Weg, – wo ist der eure?
... Den Weg nämlich – den gibt es
nicht!"

Also sprach Zarathustra

-Vom Geist der Schwere

Friedrich Nietzsche
Abstract

In order to avoid network congestion, a clear trend towards processing and analyzing an increasing amount of data at the edge of the network can be observed. Consequently, compute- and thus resource-intensive processing tasks move closer to the data source, e.g., near to the sensors. At the same time, however, the sheer number of sensors producing a huge amount of data that has to be processed in real-time raises the question of suitable compute architectures. As a result, dedicated edge platforms capable of performing such data-intensive and time-critical tasks at the edge of the network are gaining more and more interest and visibility. This work adheres to this trend yet advocates Field Programmable Gate Array (FPGA)-based System-on-Chips (SoCs), and thus both hardware- and software-programmable architectures, as ideal compute platforms to acquire and process large amounts of data right at the edge.

However, when it comes to protecting edge computing and sensitive data from potential attacks against such platforms, security must play an important role. Particularly as FPGA-based edge platforms open up attack vectors not found on traditional computing devices, due to the tight integration of programmable hardware and software components. For example, an attacker with physical access to non-volatile memory could manipulate the loading of an initial FPGA configuration, which could negatively affect the entire system. Moreover, permanent SoC memory is at risk of being attacked to obtain, e.g., cryptographic keys. Such undermining of the integrity respective confidentiality of the keys poses the most significant risk, as often the security of the entire system is based on these keys. To protect FPGA devices from the threat of key-theft and manipulation, so-called Physical Unclonable Functions (PUFs) have evolved as a promising solution for permanent key storage by converting natural transistor variations into unique, FPGA-intrinsic secrets.

This thesis investigates techniques to secure FPGA-based SoC devices at the edge against different security threats. As a first contribution, a novel digitally-tunable PUF design is presented that provides an alternative to insecure permanent key storage in an untrusted environment. Thereby, addressable shift registers available within the FPGA are used for the first time to generate PUF responses by adjustable signal propagation delays. It is demonstrated that these free-configurable propagation delays can be used to adjust a given PUF circuit to achieve better uniqueness and reliability properties.
without sacrificing the PUF’s unpredictability and unclonability characteristics. To this end, completely novel analyses and evaluations of the effects of temperature variation and stability are performed to investigate the impact of extreme temperatures on the robustness of the PUF. As a result, the proposed approach achieves significantly better results in uniqueness, reliability, and robustness than existing static PUF designs for FPGAs. Moreover, it is shown that this enables device-specific PUF configurations, which results in savings in error correction times and memory resources for cryptographic key generation.

As a second contribution, the PUF-based key generation scheme is extended to an innovative approach for secure data communication with non-volatile memory devices. Here, the reconfigurable logic of the FPGA serves as a hardware-based root of trust for security-critical authentication and integrity checking of data transfers between SoC and memory. At power-on, a hardware-protected security unit called Trusted Memory-Interface Unit (TMIU) with access to the non-volatile memory is loaded into the FPGA. Subsequently, after appropriate authentication of the memory device, the simultaneous decryption and verification of configuration data are performed, making it possible to securely boot the entire SoC. Thus, the TMIU enables for the first time to securely handle at scale FPGA-based SoC configurations as well as any other sensitive data across power cycles. Furthermore, it can be shown that the TMIU is capable to read/write to the non-volatile memory device at I/O-rate (line-rate). As a result, optimal data throughput to the storage device is maintained, even though only encrypted and verified data is loaded respectively stored.

Following the secure boot of an initial system configuration, remote firmware updates are required to ensure the secure deployment of often globally distributed edge devices throughout their entire lifetime. Especially here, immunity requirements, such as the authenticity of the update provider as well as the confidentiality and integrity of the content communicated over an untrusted network, are essential. In this context, a protocol that allows a) to uniquely verify the authorization of an external update provider and, b) based on this authentication, to establish a protected channel for secure data transfer is presented as the third contribution of this thesis. Moreover, the proposed update protocol can guarantee the confidentiality of the cryptographic keys applied throughout the update process, even if the SoC’s software is compromised. In addition, low and only temporal (i.e., during the update process) resource overheads are achieved by leveraging partial yet secure reconfiguration of cryptographic modules.

In summary, this work offers for the first time a comprehensive set of concepts, techniques, and lightweight implementations to enable lifetime security of HW/SW-programmable SoCs, starting from cryptographic key provisioning to boot protection and upgradeability.
Acknowledgments

I owe this work to an excellent scientific environment in which I was privileged to learn and conduct research. First and foremost, this is the merit of my supervisor Prof. Dr.-Ing. Jürgen Teich, to whom I would like to express my sincere thankfulness at this point. With his expectations but also with his encouragement, he has guided and supported me in the best possible way. I want to thank him, especially for his time discussing ideas, papers, and projects with me and improving them through his scientific expertise.

I also want to thank Prof. Dr.-Ing. Georg Sigl for agreeing to be the co-examiner of this work and to Prof. Dr.-Ing. Philipp Beckerle and Prof. Dr. Dominique Schröder for being part of the exam committee.

A large part of my research was funded by a doctoral research project in cooperation with the Schaeffler Technologies AG & Co. KG, in particular the Schaeffler Hub for Advanced Research at Friedrich-Alexander University (SHARE at FAU) initiative. Here, I am particularly thankful to Mr. Pschyklenk, Dr. Strohmeyer, and the SHARE at FAU for the possibility to gain insights into the research activities of a large corporation and to have worked on an industry-relevant research topic.

A huge thank you goes to all my colleagues and students who accompanied me during my time at the chair, with whom I enjoyed working, and from whom I was also able to learn so much. With Fedor, Patrick, Andreas, Christian, Behnaz, and Samu, I could not only exchange ideas about scientific topics and teaching duties but also always had a great time beyond the academic world. Special thanks go to Stefan, who over-fulfilled his role as a group leader by being a reliable advisor in all matters, who could always be relied upon. Your support, guidance, and help had a tremendous impact not only on this thesis.

Needless to say, this work would not have been possible without the everlasting support and encouragement of my family and friends. Last but not least, I must express my sincere gratitude to my girlfriend Lucy. She had probably made the most sacrifices when long working hours made things complicated. Thank you so much for your patience and understanding.
## Contents

1 Introduction .......................................................... 1
   1.1 Data Acquisition and Processing at the Edge ................. 2
   1.2 Programmable SoC (PSoC) Security: Challenges and Opportunities .... 3
   1.3 Thesis Contributions ........................................... 4
       1.3.1 Physical Unclonable Functions ............................ 5
       1.3.2 Secure PSoC Boot ......................................... 7
       1.3.3 Secure PSoC Updates ..................................... 7
   1.4 Thesis Organization ............................................. 8

2 Fundamentals ........................................................ 9
   2.1 Edge Computing and Devices .................................... 9
       2.1.1 HW/SW-Programmable SoC Architectures ................ 11
       2.1.2 Industrial Edge Computing .............................. 14
   2.2 Threat Models .................................................... 16
       2.2.1 PSoC Risk and Attack Assessment ...................... 18
       2.2.2 Principles of PSoC Protection ............................ 20
   2.3 Cryptographic Hardware Primitives ........................... 21
       2.3.1 Symmetric Cryptography .................................. 22
       2.3.2 Asymmetric Cryptography ................................. 24
       2.3.3 Cryptographic Hash Functions ............................ 28
       2.3.4 FPGA Key Storage and Device Authentication ............ 30

3 Cryptographic Key Provisioning using Physical Unclonable Functions .... 35
   3.1 Silicon Physical Unclonable Functions (PUFs) ................. 36
       3.1.1 Security Foundations ...................................... 37
       3.1.2 Process and Timing Variations ........................... 40
       3.1.3 Temperature and Aging Effects ........................... 45
       3.1.4 Related Work .............................................. 47
   3.2 CHOICE: A Signature-Tunable PUF for FPGAs .................. 51
       3.2.1 Delay Adaptation .......................................... 51
# Contents

3.2.2 Circuit Design ..................................................... 53  
3.2.3 Delay Model ......................................................... 55  
3.3 Experimental Evaluation .......................... 58  
3.3.1 Uniformity ......................................................... 60  
3.3.2 Uniqueness ......................................................... 61  
3.3.3 Reliability ......................................................... 63  
3.3.4 Temperature Robustness ............................ 64  
3.4 PUF-Based Cryptographic Key Generation ............................. 69  
3.4.1 Majority Voting .................................................... 69  
3.4.2 Fuzzy Extraction .................................................... 70  
3.4.3 Design and Evaluation of a PUF-Based Key Generator .......... 72  
3.5 Summary ................................................................. 75  

4 Booting Programmable SoCs Securely from Non-Volatile Memory 77  
4.1 Initial PSoC HW/SW Configuration .................. 78  
4.1.1 Related Work ..................................................... 82  
4.2 Hardware-Centric Secure Boot Process ............... 84  
4.2.1 Boot Process ....................................................... 85  
4.2.2 On-the-Fly Key Provisioning ......................... 87  
4.3 Trusted Memory-Interface Unit (TMIU) ................. 89  
4.3.1 Communication Interface .......................... 90  
4.3.2 Data Confidentiality and Integrity Verification .......... 91  
4.4 Results ................................................................. 93  
4.4.1 Performance Evaluation .................................. 93  
4.4.2 Hardware Resource and Power Overheads .......... 94  
4.5 Summary ................................................................. 96  

5 Updating Programmable SoCs in Untrusted Environments 99  
5.1 PSoC Remote Configuration .......................... 100  
5.1.1 System Security Model .................................. 101  
5.1.2 Related Work ..................................................... 102  
5.2 Secure Update Protocol .......................... 103  
5.2.1 Protocol ......................................................... 103  
5.2.2 Partial Reconfiguration and System Availability .......... 107  
5.2.3 Trusted Update Unit (TUU) ......................... 108  
5.3 Results ................................................................. 110  
5.3.1 Performance Evaluation .......................... 111  
5.3.2 Hardware Resource and Power Evaluation .......... 112  
5.4 Perspective Post-Quantum Security .................. 113  
5.5 Summary ................................................................. 115
## Contents

6 **Summary and Future Directions**  117
   6.1 Summary  .................................................. 117
   6.2 Future Directions ............................................. 119

A **German Part**  121

Bibliography  127

Author's Own Publications  137

Acronyms  139

List of Symbols  143

Index  147
Over the last quarter of a century, rapid advances in semiconductor and network technology have dramatically changed the way we make use of computers. Whereas in the past humans with their simple user inputs were almost exclusively the source of computational data, we now equip physical objects with hundreds of sensors to continuously generate large amounts of complex environmental data. In particular, with the advent of Cyber-Physical Systems (CPSs) and the Internet of Things (IoT), it has become possible for the first time in human history to collect real-world information on an unprecedented scale. While all along, improvements in silicon technology have kept pace with the demands of processing, storing and, transmitting data, this is becoming increasingly difficult in an age of machine-generated data that rapidly outpaces the human-generated one by several orders of magnitude [Sys20].

Today, environmental conditions such as image, sound, temperature, vibration, or acceleration data can be captured, processed, and exchanged in an entirely automated way. This data offers novel analytical methods such as sensor fusion and machine learning to improve the control and analysis of objects, plants, and processes [LKY14]. A variety of domains, such as autonomous driving, remote health, camera surveillance, and the industrial IoT, have emerged from this development but also contribute to the fact that the volume of data will continue to grow exponentially [CMZ14]. Unfortunately, the initial promise that cloud computing will cope with these data volumes could not be held [Sys18]. One reason is that on cloud servers, computation and communication goes hand in hand. As a result, processing data from more and more sensors by a single compute instance is increasingly constrained through the bandwidth of the network over which the data must be sent [SDQ10]. In addition, the lack of bandwidth guarantees makes the prediction of rigid real-time boundaries de facto impossible. However, this real-time guarantee is essential for many of the domains and applications mentioned above. Therefore, a trend towards analyzing and processing increasingly large amounts of data close to sensors can be seen [Sat17], which is why dedicated edge computing platforms for direct data processing at the network’s edge are gaining more and more interest and visibility.

1Cloud Computing [Nat1]: A model for enabling ubiquitous, on-demand network access to a pool of configurable computing resources that can be rapidly provisioned and released with minimal effort.
1.1 Data Acquisition and Processing at the Edge

A considerable challenge for suitable platforms that meet the requirements of edge computing tasks arises from the increasing number and diversity of sensors and data to be processed in real-time, which can vary substantially depending on the application [SWP+20]. Although the latest generation of CPSs and IoT devices support signal pre-processing functions such as filtering, compression, or conversions, they are often built for a single static use case and must be replaced as changes occur [SWP+20; Hig18]. Moreover, these compute platforms cannot handle heavy workloads, resulting from the simultaneous acquisition and processing of multiple sensors, respectively inputs, for which typically highly specialized analysis and Digital Signal Processor (DSP) units are used. It can be followed that not only real-time capabilities but also flexibility and scalability is required to simultaneously sense different types of signals as well as in- and outputs with the same platform.

In this context, emerging hardware- and I/O-reconfigurable Programmable System-on-Chip (PSoC) platforms based on Field Programmable Gate Arrays (FPGAs) seem ideal implementation candidates. Compared to other processing platforms, PSoCs have several advantages over, e.g., highly flexible but power-hungry general-purpose processors or fully optimized yet unflexible dedicated circuits, implemented on Application-Specific Integrated Circuits (ASICs). With their flexibility in both software (processor) and hardware (FPGA), these platforms provide the adaptability necessary for seamless data acquisition and processing for a wide range of sensors and dynamically changing data workloads. For instance, in hardware, time-critical signals can be filtered, fused, or pre-processed directly at a high sampling rate. Moreover, this can be done in parallel without putting additional load on the processor system, thus, providing scalability. Furthermore, the reconfigurable I/O ports of the FPGA enable to cover a bouquet of different sensor interfaces. At the same time, the processor allows making use of a rich ecosystem of software libraries to address the plethora of varying networking protocols available on the market. In this way, HW/SW-Programmable SoC Architectures, which this work focuses on, provide the necessary scalability and flexibility at a relatively low cost for processing vast amounts of data right at the edge of the network thus, avoiding network congestion to servers and clouds [SWP+20].

Although design tools to automatically generate optimized HW/SW applications for such heterogeneous PSoCs have been of particular interest, too, e.g., in [SLS+17; SLW+18; PSB+19], their design and implementation are not considered in this work. Instead, the thesis at hand addresses fundamental security properties of these platforms to protect the workload of edge computing tasks and enable their secure deployment in a potentially untrusted environment. In particular, regarding these platforms’ confidentiality, integrity, and availability (CIA).
1.2 Programmable SoC (PSoC) Security: Challenges and Opportunities

As the volume of data rises, so does its value, causing the edge to be increasingly seen as a potential target for cyberattacks. These threats, in turn, place the spotlight on the security of the deployed devices. For instance, while centralized cloud servers operate within well-guarded data centers, edge platforms are deployed in an environment generally considered untrusted. Primarily, due to their deployment in the field, it must be assumed that an attacker has physical access, which opens up additional attack vectors. Moreover, considering the increasing connectivity and the sheer number of sensors and actuators involved in often highly safety-critical applications, undermining security would immediately cause severe damage. This damage might include a loss of corporate reputation, exposure of confidential data, or in the worst case, even physical harm.

In this context, the main goal of an attacker is typically to access, intercept, or manipulate data stored on or transferred to a device. This may include configuration data in the form of object code for a processor system or partial or complete hardware designs in the case of an FPGA. For this reason, security standards for edge devices must rely on mandatory immunity assumptions, such as the integrity and authenticity of those initial system configurations [SFB+20]. This is especially true for FPGA-based PSoC architectures, where program codes and configuration data can reach gigabyte dimensions. For that reason, PSoC platforms load not only the software but also the configuration data for the reconfigurable FPGA structure from mass storage memory devices, such as flash or SD cards. Nevertheless, booting such initial device configurations from regular non-volatile memories is highly security-critical. In this context, attacks have already shown that an FPGA’s configuration bitstream can be manipulated so that the PSoC reveals security-relevant information [CSP+13] or even hijacks the entire system [JHZ+17]. Therefore, this thesis addresses how an initial PSoC configuration can be protected when loading from non-volatile memory in an untrusted environment without impairing system performance.

Regarding resistance, especially against attacks on software, hardware-based roots of trust deeply integrated into silicon represent the most recent stage in a progression of security measures. Applying such robust countermeasures, however, is a continuous process and cannot be specified once and for all during the design phase. For instance, applications in the avionic or industrial domain achieve operating times of 20 and even more years, which means that today’s security measures must protect against threats that may not even be known yet. In these domains, particularly long-term security poses a challenge as system security requirements arise from constantly evolving threat scenarios. This challenge can only be addressed by the often discussed concept of crypto-agility [WAG+21], which allows security mechanisms to be exchanged on demand. In this context, only FPGAs, respectively, PSoCs offer the possibility to adapt hardware security measures in the field through secure updates and, thus, remain immune against attacks.
throughout their lifetime. Addressing lifetime security, the work at hand proposes a methodology to transfer and provide a secure remote update for PSoCs in an untrusted environment.

Although it is well known that the confidentiality of data can be protected using encryption, the protection of the corresponding keys is often overlooked. As a result, FPGA vendors have offered dedicated cryptographic key storage for already many years [TM14]. Nevertheless, malicious tampering with such permanent storage up to retrieving keys remains an Achilles’ heel, as recent work has demonstrated [LTK’18; KLS’20]. Consequently, the entire trust foundation depends on the confidentiality and integrity of the secret key storage, which poses an unacceptable risk to the whole system.

An attacker who manages to read out the content of the cryptographic key storage can modify a PSoC configuration at will to load unwanted hardware functions, thieve valuable data, or execute malicious code.

From this, it can be concluded that PSoC-based edge platforms must be protected focusing on three crucial security phases, namely during (i) the secure provisioning of cryptographic keys, (ii) the boot of an initial system configuration from non-volatile memory, and (iii) the update of an existing system configuration. The work at hand contributes novel security measures to all these three phases to enable the secure deployment of HW/SW-programmable SoC platforms in an untrusted edge environment.

The results within this thesis are taken from existing publications that were published and presented in peer-reviewed journals, conferences, and workshops. In particular, the primary used publications are: [SFB’20; SWP’20; SWP’21; SKB’21a; SKB’21b]. For a complete list of the author’s publications, see Author’s Own Publications on page 137.

1.3 Thesis Contributions

As a first contribution, it was shown by the author in [Franz-Josef Streit et al. Data Acquisition and Control at the Edge: A Hardware / Software-Reconfigurable Approach. Production Engineering, 14:365–371, 3, 2020. DOI: 10.1007/s11740-020-00964-x] that PSoCs can play a crucial role in reducing the complexity of today’s automation systems as a concrete example of the need for edge computing. However, identifying suitable security concepts is essential for integrating such PSoCs in these often safety-critical applications. Within the scope of this work, necessary fundamentals concerning central aspects of PSoC security were investigated. In this realm, this dissertation presents a threefold contribution to the protection of such platforms resulting from a comprehensive risk and threat assessment.

First, a new class of signature-tunable PUFs is introduced to provide a secure alternative to vulnerable physical key storage. Here, the PUF exploits tiny process variations that naturally occur during the manufacturing process of an FPGA to create a unique and unpredictable digital fingerprint. Based on the selection of addressable-delay lines, it is demonstrated that the PUF can be tuned to achieve better security properties compared
to existing static PUF designs. Moreover, it can be shown that this PUF enables on-the-fly cryptographic key generation. In this way, keys are provided only on demand; moreover, both PUF and key are solely accessible within the reconfigurable logic of the FPGA, which avoids permanent key storage.

Since such resets or power cycles force a reinitialization of the entire system, the PUF-based key generation is integrated into a hardware-centric secure boot process for PSoCs from Non-Volatile Memory (NVM) as a second contribution. Its central component realized in reconfigurable logic is called TMIU and placed as an intermediate instance between the processor and the NVM to verify the integrity of an initial PSoC configuration.

Third, a processor-free update scheme is presented that guarantees secure remote updates even in the case the processing system of the PSoC gets compromised. Moreover, partial reconfiguration is applied during an update to keep resource overheads to a minimum and preserve the PSoC’s availability.

These novel hardware-based security measures offer several advantages over conventional software-only solutions and can build upon existing methods provided by PSoC vendors. For instance, changing security requirements during the product lifetime can be addressed by upgrading the deployed cryptographic primitives in hardware. At the same time, software vulnerabilities and performance constraints, as well as power limitations, are targeted through a dedicated implementation in reconfigurable logic. In this way, it gets possible to create a immutable yet flexible, secure system to protect proprietary intellectual property and sensitive data. Subsequently, this raises both the trust and security level of the entire device required for the acceptance of future data-driven edge applications and business models.

Figure 1.1 summarizes the three main contributions of the author that advance the system security of PSoCs. It also illustrates the structure of this thesis, which is organized according to the corresponding chapters. A brief summary of the individual contributions and chapters is given in the following sections.

1.3.1 Physical Unclonable Functions

Inevitable inaccuracies that naturally occur during the manufacturing process of Integrated Circuits (ICs) can be exploited by making use of so-called Physical Unclonable Functions (PUFs) [DGS+14]. Such PUFs have the potential to replace the current practice of non-volatile cryptographic key storage by deriving a key at runtime from a unique, silicon-intrinsic secret. However, PUFs are not yet extensively used on the current generation of ICs as they are susceptible to changing environmental conditions such as high temperature and supply voltage fluctuations. Aging effects within the transistors, which are known to be the foundation of any silicon PUF circuit, can even entirely invalidate the required security properties over time. As a solution to such PUF degradation, the work in [Franz-Josef Streit et al. CHOICE – A Tunable PUF-Design for FPGAs. In 31th International Conference on Field-Programmable Logic and Applications (FPL), pages 38–44.
IEEE, 2021. DOI: 10.1109/FPL53798.2021.00015] introduces \textit{CHOICE}, the first tunable PUF for FPGAs. The author’s main contribution is the idea to use addressable shift registers available on an FPGA to choose between individual PUF timing configurations. These configurations result in a selection of 6,144 different PUF tuning options. For the first time, this tuning makes it possible to use the same PUF to select a device-specific PUF configuration that already takes individual chip conditions such as transistor aging into account. Experimental evaluations performed by the author confirmed that this approach provides better uniqueness, reliability, and robustness properties than state-of-the-art static PUF designs. Moreover, the author proposed in this work a circuit design of \textit{CHOICE} on the widely used Xilinx Zynq platform that requires only a single FPGA slice per PUF bit. This implementation marks currently one of the smallest PUF designs available for FPGAs.

In [Franz-Josef Streit et al. Design and Evaluation of a Tunable PUF Architecture for FPGAs. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 15(1), 2021. DOI: 10.1145/3491237], the author extended his work in [SKB+21a] by comprehensively investigating how temperature variations affect the properties of the PUF response. Here, experimental evaluations revealed that due to \textit{CHOICE}’s fine-tuning capabilities, configurations could be found that provide exceptional robustness even to extreme temperature variations. Thus, the author could demonstrate that the device-specific PUF-signatures of \textit{CHOICE} outperform their static counterparts even under harsh environmental conditions. To highlight the practicability of the approach, a case study was
presented that showcases the savings in terms of performance and hardware resources when using CHOICE for cryptographic key generation within an extensive temperature range of $\Delta T = 70$ K. Detailed results and contributions are described in Chapter 3.

### 1.3.2 Secure PSoC Boot

The trustworthy initialization of PSoC platforms after power-on is only possible if the integrity and authenticity of the initial hardware and software configuration, loaded from Non-Volatile Memory (NVM), can be fully verified during the boot phase. The author contributes to this effort with a novel multi-stage verification process for secure booting from external NVM exploiting the FPGA’s reconfigurable logic, as first proposed in [Franz-Josef Streit et al. Secure Boot from Non-Volatile Memory for Programmable SoC Architectures. In 13th International Symposium on Hardware-Oriented Security and Trust (HOST), pages 102–110. IEEE, 2020. DOI: 10.1109/HOST45689.2020.9300126 ]. Here, the FPGA serves as a secure anchor point by performing the necessary integrity and authenticity checks for secure data communication with non-volatile memory devices.

To this end, a hardware design called Trusted Memory-Interface Unit (TMIU) is loaded at boot time in a first step to ensure the authenticity of an encrypted NVM and the secure decryption and verification of its content in a second. The main contribution of this methodology is that, together with the PUF-based key generation from Section 1.3.1, an innovative approach for secure data communication with non-volatile memory devices is achieved. In fact, with the TMIU, it becomes possible to securely manage both PSoC configurations and other sensitive data at a large scale across restart cycles. Moreover, thanks to the PUF-based key provisioning, the coupling of NVM content to a specific PSoC device gets possible without relying on external key storage. In this way, the TMIU locks the NVM content to the PSoC, forming a permanent and immutable system that protects proprietary Intellectual Property (IP) and sensitive data. Consequently, a configuration can be booted from NVM only if it is successfully authenticated and found in an untampered condition. Finally, this verification process increases trust and security in the entire system and allows a system designer to keep track of delivered IP.

An investigation regarding the power and resource overheads of the proposed protection mechanisms showed that the TMIU implementation could already be used in relatively small, even IoT devices. Moreover, it was demonstrated that the proposed methodology does not limit the performance of a Xilinx Zynq PSoC boot process, as the computation required for integrity verification can be performed at memory I/O-rate (line-rate).

All the above contributions will be described in detail in Chapter 4.

### 1.3.3 Secure PSoC Updates

The essential need to prevent unauthorized access to configuration data does not end with system boot. It follows that runtime security must also be ensured by design to
guarantee the secure deployment of PSoC platforms throughout their lifetime. For this reason, a methodology for the secure transfer and provision of hardware/software updates was proposed by the author in [Franz-Josef Streit et al. Providing Tamper-Secure SoC Updates Through Reconfigurable Hardware. In 17th International Symposium on Applied Reconfigurable Computing (ARC), pages 242–253. Springer, 2021. DOI: 10.1007/978-3-030-79025-7_17 ]. In this work, the author’s contribution includes the presentation of a hardware design called *Trusted Update Unit (TUU)*, which is loaded into the partially reconfigurable area of the FPGA for the time of an update.

The TUU enables performing a cryptographic update protocol entirely in hardware by verifying the identity of an external update source and then, based on this validation, provides a secure channel for protected data transfers to the PSoC. In addition, an enhanced security model is formulated by the author that dictates that the PSoC can only be reprogrammed via a secure remote update or reboot session. Moreover, the model requires that secret keys shall at any time not be accessible from the PSoC’s CPU processing system. In this way, the integrity and confidentiality of the keys can be guaranteed by leveraging security-sensitive update operations entirely within the programmable logic of the PSoC. Since hardware resource requirements for such asymmetric cryptographic schemes are typically large, the author’s idea is to integrate a partial reconfiguration flow into the update protocol. As a result, hardware resource utilization occurs only temporal, i.e., during the time of an update, while still adhering with the enhanced security model mentioned before. The author also presents solutions to central security concerns, such as how system availability is maintained and how partial or complete updates can be handled, including the crypto primitives themselves. An in-depth introduction of the remote update protocol is presented in Chapter 5.

### 1.4 Thesis Organization

The remainder of this dissertation is organized as follows: Chapter 2 lays out the fundamental concepts required for this work, including an introduction to edge computing, a detailed system description of HW/SW-Programmable SoC architectures, a comprehensive attack and risk assessment of these platforms, and the fundamentals of the cryptographic algorithms and protection principles used in this thesis. Chapter 3 introduces the concept of silicon PUFs and PUF-based cryptographic key generation as an alternative to physical key storage based on the signature-tunable CHOICE PUF along with its configuration options. The requirements for securely loading an initial hardware/software configuration for PSoCs from non-volatile memory are discussed in Chapter 4 before presenting the proposed multi-stage secure boot process together with the TMIU for protected memory transfers. Subsequently, Chapter 5 addresses the topic of remote PSoC updates over an untrusted network by presenting a secure update protocol entirely performed in hardware. Finally, Chapter 6 briefly summarizes the findings of this thesis and concludes with an outlook on possible future directions.
This chapter presents the fundamentals and requirements of the concepts that form the basis for this work. Since this thesis is concerned with techniques to secure the operation of PSoCs in the context of edge computing, an exemplary system security model for such platforms is presented. The system security model consists of four sequential layers, shown in Figure 2.1, with each layer building on the next to provide a guideline for adequate security measures. The system’s intended use, function, and architecture is defined at the first layer. Before presenting a concrete use case of industrial edge computing on a PSoC-based edge architecture, the general idea of edge computing and its devices is introduced in Section 2.1. This includes the requirements that such devices must meet at the edge, followed by a discussion of the architecture-specific benefits of PSoCs\(^1\). Subsequently, Section 2.2 explains potential risk scenarios associated with the second layer and describes the mandatory protection goals specific to these SoCs, which form the third layer of the model. Finally, Section 2.3 introduces the basic cryptographic principles underlying the security solutions proposed in this thesis, thereby forming the last layer. This includes various design considerations when implementing the solutions in hardware and a comparison with tamper protection measures provided by FPGA/PSoC vendors.

### 2.1 Edge Computing and Devices

According to the Cisco Global Cloud Index 2016-2021 [Sys18], data stored in data centers worldwide will reach 1.3 ZB by the end of 2021. However, at the same time, the total amount of data generated by Internet-enabled devices will be 847 ZB. This progression shows already today that the data generated exceeds the data that can be transferred and stored by almost two orders of magnitude. In fact, this development will continue in the future, as predicted by the Cisco Annual Internet Report [Sys20]. According to this report, by 2023, the number of networked devices will reach 29.3 billion. This is mainly driven by the IoT and its fully automated machine-to-machine connections, whose

\(^1\)The content of this section is partially based on and published in [SWP’20] which has appeared in the Journal of Production Engineering.
share will increase from 33 percent in 2018 to 50 percent by 2023. Many of these IoT applications generate a large amount of data but at the same time require short response times, which imposes tremendous pressure on the networks [SCZ16]. Therefore, in a post-cloud era, edge computing aims, on the one hand, to reduce latency but, on the other, reduce the cost of the data transport by processing large quantities of this data directly at the edge of the network. In this line, edge computing is defined in a broader sense as:

**Edge Computing** [SCZ16]: Refers to the enabling technologies allowing computation to be performed at the edge of the network, on downstream data on behalf of cloud services and upstream data on behalf of IoT services.

The shift of data traffic from the core of the network to the edge has a particular impact on the computing and communications architectures required to process this
data. As a result, both academia and industry are discussing how future computing architectures should look like to handle the sheer amount of data, which has led to the emergence of dedicated edge devices.

**Edge Devices**  The general idea of an edge device is to transmit data packets between different network nodes in interconnected structures, be it the IoT or any other network of physical objects. Unlike traditional routers, switches, or gateways, however, these devices additionally process the data generated by sensors and actuators from areas such as autonomous driving, medical signal processing, or industrial automation directly at the edge of the network. This pre-processing allows extensive filtering or an in-situ analysis at the so-called field level (sensors/actuators), reducing masses of data generated by more and more sensors. As a result, this makes the control, management, and processing of such sensor information much more efficient, while reducing traffic and network load.

Let’s take the computing requirements in industrial automation systems as an example (cf. Section 2.1.2). In this area, it becomes clear that processing data close to the sensor instead of processing it on central powerful compute units brings new challenges. In fact, the idea of outsourcing time-consuming and costly processing to the field level to avoid sending large volumes of data to the cloud also requires significant computational capacity. A Programmable System-on-Chip (PSoC)’s intended use and function in these industrial areas form the foundation of the first layer in the proposed system security model (cf. Figure 2.1). In the following, a detailed overview of a typical PSoC architecture is given before discussing PSoC-specific threats and protection mechanisms in the second and third layers. It is precisely such architectures that allow achieving the required flexibility and scalability for edge computing in an industrial context, taking advantage of hardware reconfiguration and hardware acceleration.

### 2.1 HW/SW-Programmable SoC Architectures

**Hardware/Software-Programmable System-on-Chips (PSoCs)** are hybrid systems that integrate Field Programmable Gate Array (FPGA) logic with one or even more microprocessors, memory blocks, peripherals, and external interfaces on a single chip. In this thesis, PSoCs are considered where an SRAM-based FPGA is tightly coupled to a multicore processor system via a central interconnect, see also Figure 2.2. Here, both the processor system and the FPGA have access to external main memory (DDR) to which data is transferred after a successful boot from a Non-Volatile Memory (NVM) storage device that is also external to the chip. In addition, the processor system supports a general-purpose operating system (e.g., Linux) to provide higher-level task processing as well as commonly needed network and cloud connectivity. The reconfigurable FPGA hardware can be used for data acquisition and control of sensors and actors but could also take over security-critical operations. Compared to alternatives such as DSPs and
AsICs, such FPGA-based systems combine the advantage of low latency through dedicated hardware and, at the same time, enable the use of a wide variety of already existing software libraries, as detailed next.

Field Programmable Gate Arrays (FPGAs). FPGAs belong to the family of logic-programmable devices, where pre-fabricated programmable cells and interconnects make it possible to design digital hardware modules for any application. Different to CPUs, FPGAs are not programmed by instruction sequences, but rather spatially: A hardware circuit is implemented typically by loading a stream of configuration bits into the device that customizes the logic functions of the pre-fabricated cells and their interconnect. Such hardware circuits can not only compute a function internally in parallel, but also multiple instances of the circuit can be executed in parallel. This concurrency allows performance gains from the simultaneous computations of the designed circuits compared
Figure 2.3: Schematic illustration of the construction of a Xilinx FPGA with its fundamental building blocks of so-called CLBs, reconfigurable interconnects, and in-/outputs (I/O). The detailed view on the right illustrates the atomic FPGA resources within a CLB, namely Lookup Tables (LUTs), Flip-Flops (FFs), and the carry chain.

to the often sequential execution of object code on a general-purpose processor. To accomplish this, a digital circuit of logic gates is mapped onto the FPGA, which is typically arranged as a two-dimensional matrix of hard macros, so-called Configurable Logic Blocks (CLBs). Each CLB is connected via a switch matrix to a general reconfigurable interconnect called routing matrix.

A schematic representation of a small subset of such programmable logic of a Xilinx FPGA, together with its reconfigurable routing matrix and in-/outputs (I/O), is shown in Figure 2.3. On Xilinx devices, each such CLB contains two slices (shown in the right half of Figure 2.3), with one slice consisting of four 6-input LUTs LUT0, ..., LUT3, eight FFs FF0, ..., FF7, a wide-function multiplexer carry logic in the form of an arithmetic carry chain (CARRY4), and programmable wires (not shown in the right half of Figure 2.3). This allows, for example, a 6-input LUT to be used to implement the representation of any Boolean function with up to 6 inputs. The FPGA-fabric can also implement any Boolean function with more than 6 inputs by connecting these LUTs via the reconfigurable interconnect between them. Since these LUTs are built from SRAM cells, the FPGA must be reconfigured after each power-on cycle by loading a generated bit file that fills the LUTs and defines the interconnect configurations accordingly.

Xilinx distinguishes between two different types of CLB slices. In addition to the logic slices already mentioned, memory slices are offered that allow certain LUTs to be configured as distributed memory or as Addressable Shift Register (ASR) logic. While these special LUTs can either be used for small storage of up to 64 bits or as ASRs, allowing the implementation of shift registers with a variable length of 32 different stages. Besides
distributed memory based on these LUTs, Xilinx FPGAs feature dedicated memory blocks in the form of 36 kbit Block RAM (BRAM) cells, as well as dedicated DSP slices to implement accumulators and multipliers. To interface with the outside world, the programmable logic blocks are surrounded by numerous input/output (I/O) ports (cf. Figure 2.3) that can equally be configured to the need of an application. Contrary to the often open-source library concepts of programming languages, however, functionalities and more complex designs for FPGAs are provided by Intellectual Property (IP) cores, which are highly valuable and, therefore, usually proprietary. Hence, the confidentiality and integrity of IP and bitstreams must be protected when they are communicated over insecure information channels or stored in configuration memories, as discussed in more detail in Section 2.2.

To conclude, hardware/software-reconfigurable SoCs combine the convenience of programmability with powerful data processing through reconfigurable hardware. In this regard, the programmable logic within the PSoC allows it to meet potentially changing requirements during the lifetime of an edge device, as its configuration can be adapted to new conditions through reprogramming. As an example, the following subsection proposes the application of such PSoCs for an industrial edge computing use case.

2.1.2 Industrial Edge Computing

Today’s manufacturing plants and processes offer the potential to collect data on an unprecedented scale. For this reason, industrial applications are increasingly being equipped with edge devices that enable the required connectivity and gateway function to transmit the data to higher-level instances, such as the cloud. Especially in an industrial setting, this is often accompanied by data acquisition, processing, and control tasks from various sensors, as well as fieldbuses or other automation protocols. Therefore, an ideal platform for industrial edge computing must provide high flexibility combined with sufficient computing power to allow the simultaneous processing of field and control level tasks close to the edge.

However, automation systems typically used in this context, such as Programmable Logic Controllers (PLCs), leave a great deal of data processing potential unused. As they are not designed to handle the seamless acquisition and processing of enormous amounts of data, the field and control levels need to be separated completely. Furthermore, their strong focus on simple control tasks and a rigid number of static built-in input/output ports make them unsuitable for the challenge of “Industrial Big Data” in a rapidly changing environment. This inflexibility is in contrast to the fact that different production machines often have specific requirements regarding the number of inputs and outputs, as well as their type, which could only be addressed by a modular approach that enables machine-specific solutions. Moreover, as operating times of up to twenty years are common in such industrial automation systems, flexibility is vital to retrofit old production plants. However, existing machines can only be cost-effectively expanded,
respectively, retrofitted with functionality if the processing system allows reconfiguration of software and hardware. In this context, FPGA-based PSoCs and, thus, hardware- and software-reconfigurable platforms seem to be a promising solution, as shown next.

**PLCs Based on Hardware and Software Reconfigurable PSoCs**

The concept of a PSoC-based PLC platform for industrial automation systems is exemplified by consulting Figure 2.2 again. Here, the processor system is used to orchestrate higher-level tasks in software, including process visualization or providing the necessary cloud and gateway connectivity, which plays a critical role in evaluating the practicability to serve in an industrial environment. In addition, the processor system can exchange data and control commands with the programmable logic area of the FPGA via the central interconnect, with the solid arrows indicating the possible data flow and the dashed arrows representing eventual control flow. Such control might be desirable to adjust parameters, e.g., for filter coefficients or the window function in signal processing algorithms on-demand at runtime.

As the type and number of signals connected to the PSoC-based PLC might also change, the FPGA’s I/O-Interconnect can be reprogrammed as inputs, outputs, or even bidirectional. This **I/O reconfigurability** of hardware interfaces makes it possible to cover not only digital and analog signals but also to implement various standard industrial fieldbus interfaces and protocols, such as Modbus, EtherCAT, or Controller Area Network (CAN) on the same ports if required. This way, application-specific actuators, and sensors can be attached and exchanged as needed. This flexibility is illustrated in Figure 2.2 for a number $I$ of different sensors, including the corresponding information and data flow. Here, sensor data is acquired via sensor-specific I/O interfaces $I/O_0, \ldots, I/O_{I-1}$ and directly passed to the FPGA for further processing. This data generated by the sensors could cause an unacceptable computational overhead to the network and the CPU processing system. Therefore, computationally intensive processing, such as signal filtering, fusion, or data conversion, is directly performed in hardware. By loading dedicated hardware IP modules, shown in Figure 2.2 as HW IP$_0, \ldots, $HW IP$_{I-1}$, the simultaneous processing of the corresponding sensors Sensor$_0, \ldots, $Sensor$_{I-1}$ is possible in parallel, which consequently relieves networks and software processing systems. Here, the pre-processing modules’ number and their computational complexity determine the amount of logic resources needed to map this functionality to hardware. As these resource requirements are specified by the amount of required FFs, LUTs, BRAMs, and DSP slices mentioned above, only the available hardware resources on a given FPGA limit the data processing capacity. Thus, it can be said that PSoC-based PLC platforms enable deterministic, in-situ signal processing for industrial applications with changing data loads even in time-critical control applications. Nevertheless, the design and implementation of applications on such heterogeneous PSoC architectures still requires an in-depth expertise in hardware/software co-design. Therefore, model-based design
approaches to automatically generate optimized HW/SW configurations for a given
PSoC are also subject of current research efforts [SLS+17; SLW+18; PSB+19].

In summary, FPGA-based PSoCs can meet the complex requirements of future in-
dustrial automation systems by providing an optimal basis to handle edge processing
tasks of the field, control, as well as higher supervisory levels by the same cost-effective
platform. However, due to their connectivity and programmability and the fact that
they are often used in a highly safety-critical environment, measures must be taken to
protect against attacks targeting these platforms. Thus, the following sections discuss
possible attack scenarios and compelling security goals when deploying such platforms
in the field.

2.2 Threat Models

On PSoCs, security is a cross-cutting concern that needs to be addressed at the software,
the network, and the hardware (physical) layer. Therefore, this section first establishes
in Section 2.2.1 a taxonomy of threat scenarios, including an overview of selected attacks
for each of the mentioned PSoC layers. This introductory remark is followed by an
overview of possible countermeasures in Section 2.2.2.

It is no surprise that the security of a digital system is always imperfect. Any security
measures can be broken, depending on an attacker’s time, money, and resources. There-
fore, part of threat modeling is to classify attacks by their prerequisites to assess the
threat of a system as realistically as possible. Here, the threat analysis of PSoCs differs
not much from that of other embedded devices, where an attacker can gain physical
access. One difference, however, is the close interaction between hardware and software,
which in turn can have an impact on their mutual security. In this context, attacks are
classified according to the three core security principles of Confidentiality, Integrity, and
Availability (CIA) they threaten [RRK+04; Zie00]. These attacks often target the resources
of PSoCs that are of high value, so-called security assets (definition in Tec09). Here,
attacks that violate the confidentiality of the PSoC can result in the theft of sensitive
data such as keys, processor object code, hardware IP, or authentication and user infor-
mation. Although copyright infringement is not an attack in the conventional sense, it is
equally a breach because it threatens the secrecy of valuable IP. Attacks that compromise
integrity can be divided into data manipulation, tampering with software or hardware,
and forgery of authorship, e.g., to gain control access to a system, while attacks that can
render a system inoperable compromise availability.

To protect these assets, it is necessary to define possible attack vectors that describe
the means to launch an attack. Therefore, another way to classify attacks can be done
according to how the attack is carried out. Figure 2.4 provides an overview of the
two main attack vectors of a PSoC device that have been most frequently exploited in
other embedded systems [RRK+04; Zie00]. Here, the means of launching the attack or
the way in which the attacks are carried out can be divided into physical and logical
attacks. While physical attacks cover methods of physical exploitation and directly target the hardware, logical attacks often exploit weaknesses and vulnerabilities in the implementation of the device. These include vulnerabilities in software and network or communication protocols, so they require relatively little effort and resources; often, a simple computer and a network connection to the attack target are all that is needed. Section 2.2.1 briefly discusses specific vulnerabilities where logical attacks have been exploited in the past.

Physical attacks have their name as they require physical access to an embedded device to extract the assets and values protected by the device [RA19]. Unlike logical attacks, physical attacks typically require additional equipment, infrastructure, and expertise. Both groups can be further subdivided into invasive or non-invasive attacks.

In particular, the subcategories of physical attacks are of interest for this thesis and are described by the work of Rahman et al. [RST+18] as follows: Non-invasive denote attacks that leave no evidence of tampering, as these attacks are accomplished without damaging the device itself. In contrast, invasive attacks involve altering or damaging the internal components, such as removing the package and revealing internal structures of the chip to obtain information. In between these two types of attacks, semi-invasive attacks are worth to mention. This category often involves optical instruments that allow optical probing or fault injection. Here, direct contact with metal layers is usually not required, so the internal structure of the semiconductor remains functional. These types of attacks pose a more significant threat to PSoC security because they require less time and expertise than invasive attacks. Because valuable information and assets must be recovered across power cycles, especially on PSoCs, all types of non-volatile memories are threatened by such attacks. In addition to non-volatile-memory mass storage devices, this includes cryptographic key storage, such as programmable read-only memories or one-time programmable eFuse registers. A more in-depth discussion is provided in Section 2.3.4.

Based on this classification, the following subsections present a selection of concrete attacks targeting PSoC-based systems and their hardware, as well as a catalog of possible (counter-) measures to avoid or prevent such attacks.
2.2.1 PSoC Risk and Attack Assessment

The risk assessment approach is to estimate the likelihood of security attacks and their consequences for a given system. Therefore, the following subsection will present a selection of concrete attack possibilities and the resulting threats specific to PSoC architectures. In this context, it should be noted that different attacks can exploit different security objectives. For this reason, following the classification of attack vectors made previously, a distinction is made between attacks on software, network communication, and physical attacks against hardware. This is particularly important regarding the co-existence of the processor and the FPGA that share the same bus system (cf. Figure 2.2 in Section 2.1.1) in a PSoC. Here, both software attacks and compromised hardware designs jeopardize the entire system’s security. Furthermore, network connectivity to the cloud and other higher-level entities further increase the attack surface.

Software Attacks

At the software level, rootkits, viruses, and code injection, as part of logical attacks, jeopardize system security when, for example, they undermine memory protection [CSF+20] or when they allow to extract secret keys using, e.g., cache attacks [RGG+19]. In addition, buffer over-reads to access adjacent memory regions [Som16] and open side-channels can expose sensible timing information [CHV+03], which successfully leaked secret passwords, cryptographic keys, login credentials, and other private data in the past. One way to perform such an attack is to inject malicious code into the system and then alter the control flow to jump to their memory location for triggering. Attacks based on code injection or buffer overflows could have dramatic consequences in an industrial setting. In this context, malicious software and hardware infiltration is becoming a serious threat since a design is often not developed from scratch to save time and costs. Instead, parts of the design are purchased from third-party vendors whose complete internal structure is often unknown, leaving room for malicious functionality insertion.

Network Attacks

Networked edge and IoT devices, including PSoCs platforms, are mainly controlled remotely via well-known software protocols such as Transport Layer Security (TLS) and its predecessor Secure Sockets Layer (SSL) [ZSA+19]. In the past, however, these protocols have been heavily affected by numerous cryptographic breaks, often due to hidden flaws in the software implementation. These flaws caused, among other weaknesses, vulnerabilities that affect the system’s availability by so-called denial-of-service attacks, as well as, vulnerabilities that exploit an insufficient authentication. In this context, particularly, man-in-the-middle attacks where an attacker gains access to the connection medium and, thus, control of the communication channel can have disastrous consequences.
2.2 Threat Models

**Hardware Attacks**

A critical security objective when using PSoCs tightly coupled with SRAM-based programmable logic is the configuration bitstream typically loaded from non-volatile memory after power-on. In this context, tampering attacks aimed at manipulating the bitstream pose the greatest threat to the use of FPGA-based PSoCs. This threat mainly concerns the transfer of the hardware configuration to the FPGA since, e.g., malicious manipulation is possible by loading from insecure sources. As part of invasive or semi-invasive attacks, such attacks aim to tamper with the integrity of the bitstream stored on non-volatile memory. In doing so, an attacker can manipulate a bitstream to cause undesired or illegal behavior of the system. A form of non-invasive attack is if an untrusted developer adds additional malicious circuitry to a hardware design, a so-called hardware Trojan horse [TK10]. Such attacks could, for instance, be performed during the manufacturing or deployment phase of the device. A Trojan might then even be able to directly perform bus or memory manipulations or load an unauthenticated PSoC configuration from an external network source. The authors in [JHZ+17] demonstrated this form of attack to break the device’s secure boot process.

Attacks that aim to clone, overbuild, or reverse engineer a hardware design initially only jeopardize the confidentiality of a bitstream or a PSoC configuration and thus, do not pose an immediate threat to the intended system operation. Nevertheless, this type of attack can result in considerable financial damage since technological advancement that may have been built up over many years is quickly caught up.

Finally, the danger of physical attacks in the form of non- or semi-invasive side-channel attacks should be pointed out. Side-channels are digital signal sources of a system or cryptographic primitive that reveal information ranging from their implementation to keys and other confidential data. These signal sources include timing variance, temperature, chip power dissipation, and even electromagnetic radiation [Sta10]. Attackers have developed techniques to observe such side-channels and draw conclusions about the system’s internals from recorded traces of these channels. In this context, specific encryption algorithms or the key storage themselves can be considered central targets for an attack. For example, Moradi et al. [MBK+11; MKP12] described a side-channel attack that uses Differential Power Analysis (DPA) to reveal the secret key of the FPGA vendor bitstream decryption module. Other semi-invasive physical attacks that directly target the integrity of the key storage to obtain confidential keys are discussed in Section 2.3.4.

Existing security techniques from the conventional processor world can only provide limited protection against intrusion as a defense against physical attacks, and manipulation must also be considered for PSoC-based edge devices in an untrusted environment. Among these, attacks that violate the integrity of a system by attacking keys pose the biggest threat to PSoC security. Despite their considerable effort, attacks targeting cryptographic keys must be considered highly critical, which is why strong protection measures must be established, as discussed next.
2.2.2 Principles of PSoC Protection

Similar to the threats and attacks already mentioned, suitable protection mechanisms can be applied or combined at different system abstraction levels. In this context, however, it is not advisable to decompose a given PSoC platform into its hardware, software, and network components and subsequently find appropriate countermeasures for each attack vector. Instead, it is more advisable to establish protection measures for the processes where the system is most vulnerable. This is particularly true in the view of adversaries with physical access since it must be assumed that they might be able to access internal data through physical attacks. This means, equally, that the attacker can obtain keys via side-channels, and bypass any physical protective measures taken. Therefore, the security concepts proposed in this work aim to extend existing solutions by explicitly focusing on protecting keys and PSoC configurations during particular critical operations. In doing so, these operations can be reduced to three particularly critical phases of PSoC security:

- First, during cryptographic key storage and provisioning;
- Second, when loading an initial system configuration from non-volatile memory;
- Third, when updating a system configuration from a remote update provider.

If these three critical phases are protected via a security-oriented system architecture, severe damage from potential attacks can be avoided. However, to not interfere with the device’s intended function, the security architecture also needs to address critical non-functional properties such as performance, resource and energy requirements, but also flexibility. Balancing the need for high performance on the one hand and sophisticated protection measures on the other is a challenge that is difficult to overcome with conventional computer architectures. For example, it is difficult for purely software-based systems to provide strict isolation of security-critical data from the user’s application code because general-purpose processors typically have a single physical memory that holds programs and data simultaneously. In contrast, FPGAs offer several advantages over conventional software-based systems by having a much smaller and better-defined attack surface in addition to their inherent hardware performance.

FPGA-based Hardware Root of Trust

FPGAs have been proven to provide all necessary means for protecting security-critical applications as long as it is guaranteed that their own configuration is provided securely [PCN11]. For instance, their dedicated logic and memory blocks offer advanced protection of security-critical operations by isolating them in hardware. This is possible because a cryptographic hardware function exists exclusively within their corresponding configuration of LUTs, FFs, and routing matrices (cf. Figure 2.3). Consequently, this isolation from other application tasks hosted by the system makes it significantly
more challenging to attack security-critical operations. Therefore, to protect the first critical phase of PSoC security, this thesis proposes a method to generate on-demand cryptographic keys within the FPGA’s logic to minimize the risk of key theft. This hardware-intrinsic key generation will be presented in detail in Chapter 3.

However, since SRAM-based FPGAs themselves must be configured by loading a bitstream stored in an external non-volatile memory after each power cycle, they rely heavily on the integrity and confidentiality of these initial configurations. For this reason, bitstream encryption is required to protect the design and IP from attempts to be cloned [SZT08] or reverse-engineered [Tri07] to meet confidentiality requirements. Nevertheless, only the combination of bitstream encryption and authentication can prevent the risk of malicious tampering. In this way, the integrity of the FPGA is simultaneously protected and verified since it can be proven that the bitstream originates from a trusted entity. As a result, an FPGA can provide a flexible, independent, and in addition long-term trust basis, a so-called hardware root of trust, for various security-critical operations within the system. This motivates the second critical phase of PSoC security, the secure loading of such a configuration during boot, called secure boot.

A secure boot process ensures that only a boot image, uniquely authenticable by a signature, is loaded. In consequence, the PSoC receives an intended, genuine configuration after power-on. In this context, earlier work has shown [PCS18] that such secure boot processes can be entirely carried out within the FPGA’s programmable hardware. In this way, the FPGA is in charge of loading the configuration from non-volatile memory. Moreover, the FPGA creates a chain of trust by performing security-critical integrity and authenticity checks in hardware. Such a method of hardware-oriented secure booting from non-volatile memory that extends related work is presented in Chapter 4.

The last critical phase of PSoC security is providing and protecting remote updates and upgrade functions to guarantee the PSoC’s security over its entire lifetime. This phase is crucial as it is required to protect the configuration of a PSoC against malicious network sources. In this context, secure FPGA update protocols have been developed, e.g., in [DK09] to defend network attacks. However, these are often based on sharing a symmetric key between a server instance and the fielded device to keep hardware requirements minimal. Therefore, a way to enable such secure remote updating through an FPGA-based root of trust while holding hardware resource requirements to a minimum is presented in Chapter 5.

Before discussing in more depth the three critical phases of PSoC security in the individual chapters, the following section first introduces the basic cryptographic operations underlying the proposed protection methods.

### 2.3 Cryptographic Hardware Primitives

Most security solutions rely on active protection measures, e.g., cryptographic precautions. Other measures include reactive actions such as detecting (physical) tampering
attempts by special tamper-sensing sensors to protect against invasive and semi-invasive attacks [RRC04]. This section focuses on the active measures and gives a brief overview of the fundamental cryptographic techniques and their notations used throughout this work. In addition, major design decisions that must be addressed when implementing these techniques in reprogrammable logic are explained. For this purpose, the used primitives for symmetric and asymmetric cryptography are first presented in Section 2.3.1, respectively Section 2.3.2, introducing their security level and key length. The applied cryptographic hash functions follow in Section 2.3.3. Subsequently, in Section 2.3.4, the protection mechanisms provided by FPGA, respectively PSoC, vendors are compared regarding their options for secure key storage and device authentication. Possible threats against physical attacks on these options are explained as well.

2.3.1 Symmetric Cryptography

The central concept of symmetric cryptography is the use of an identical key for operations such as message encryption (confidentiality), integrity verification (integrity), and authentication (authenticity). For encryption, a sender encrypts a message with a key before sending that message to a receiver. The recipient can only decrypt the received message if the same key is applied for decryption, previously used for encryption. This key ensures that only communicating parties in possession of the key can read messages. The same principle applies to the integrity and authentication validation of messages, which are not necessarily encrypted. Here, a so-called Message Authentication Code (MAC) irrevocably links a message with the shared key and calculates a digest over this data. This digest is then passed along with the message to a receiver to authenticate its sender. If the recipient holds the same key, he or she can verify both the sender and the integrity of the message if the calculated digest is matching. Therefore, a digest can be seen as a digital signature of the message, even if this notion is exclusively reserved for signature algorithms based on asymmetric keys, see Section 2.3.2. Digests are usually calculated based on a cryptographic hash function; the concept of hash functions is detailed in Section 2.3.3.

Symmetric AES Ciphers

In this work, the Advanced Encryption Standard (AES) [DV98] algorithm is used to perform symmetric en- and decryption to ensure data confidentiality. AES was standardized by the National Institute of Standards and Technology (NIST) in 2001 [Nat01] as a successor of the depreciated Triple Data Encryption Algorithm (TDEA) scheme and has since become the most common algorithm for block cipher data encryption. Compared to more recent rather lightweight symmetric encryption algorithms, such as SIMON/SPECK [BSS+15] or PRESENT [BKL+07], AES was chosen in this thesis for the following two reasons. First, a finer selection of the security level is possible by choosing between three standardized key sizes, \( Z \) of either 128-bit, 192-bit, or 256-bit. And second,
AES has undergone more extensive investigation over the years concerning possible side-channels attacks.

Throughout this work, a symmetric AES operation is denoted as a function $\epsilon (D, k_s)$ that encrypts a set of input data elements $D$ into a set of encrypted output data elements $E$, given a symmetric secret key $k_s \in \{0, 1\}^Z$ of length $Z$. This process is illustrated in Figure 2.5a of Figure 2.5 by the most computationally simple of all AES block cipher modes, the ECB mode. Here, a data input block $D_i$ is mapped directly to an encrypted data block $E_i$. This one-to-one mapping creates a highly insecure behavior of encrypting two identical data input blocks, e.g., $D_i \equiv D_j$, to two identical output blocks $E_i \equiv E_j$.

A more sophisticated alternative to the ECB mode is the CBC cipher mode, as shown in Figure 2.5b, which is solely applied throughout this work. In CBC, a previously encrypted cipher $E_{i-1}$ is XOR-linked with the subsequent input data block $D_i$ for encryption. In this way, each cipher block $E_i$ depends on its predecessor down to the very first input block $D_0$. Nevertheless, to guarantee that each encrypted message is unique, the very first block must be XOR-linked too, which requires an additional initialization vector $IV$. As a result, it gets more difficult for an attacker to learn the key from a chosen
plaintext and corresponding ciphertext. On the downside, it also complicates the AES usage due to the handling of different IVs for independent messages. For instance, in the proposed secure boot, respectively, update scheme of Chapter 4 and Chapter 5, sector-wise, respectively, update-wise de- and encryption is conducted based on the CBC mode. To close the door for possible attacks, the provisioning of initialization vectors is in these scenarios not performed by, e.g., directly using the sector number for non-volatile memory encryption. Instead, each sector number corresponds to an individual MAC tag, acting as the IV.

While AES decrypts or encrypts a message always by operating on a number of 128-bit input data blocks $D_0, \ldots, D_{i-1}$, the rounds, and thus, the time required to process these input blocks depend on the selected key size. In this context, the security level increases with a larger key size $Z$ but at the price of a higher computational effort. It holds a secret AES key $k_s \in \{0, 1\}^Z$ with $Z = 128$-bit leads to 10 rounds, $Z = 192$ to 12 rounds, and $Z = 256$ to 14 rounds. Thanks to the reconfigurable nature of FPGAs, the key lengths of an AES module implemented in hardware can be adjusted as performance permits. After all, the security of AES relies on the assumption that an attacker with access to the encrypted data cannot guess the secret key $k_s \in \{0, 1\}^Z$ without performing infeasible amounts of computation.

### 2.3.2 Asymmetric Cryptography

Unlike symmetric cryptographic operations, asymmetric cryptography requires a pair of keys that correspond to each other but are not the same. This key pair usually consists of a secret, respectively, private key denoted throughout this work as $k_s \in \{0, 1\}^Z$ of length $Z$, and additionally of a public key, denoted in the following as $k_p \in \{0, 1\}^Z$ of length $Z$. While the public key, as the name implies, can be known by the public without compromising security, the private key is known only to the owner and must be kept secret. Nevertheless, both keys are needed to perform an asymmetric cryptographic scheme, e.g., for digitally signing a message, asymmetric encryption, or key exchange protocols to derive a shared secret for symmetric data encryption.

**A brief recap of asymmetric cryptography:** Each communicating partner can encrypt messages with the public key $k_p \in \{0, 1\}^Z$ of an intended recipient. To decrypt the message, however, the corresponding secret key $k_s \in \{0, 1\}^Z$ must be used to get back the original message. This way, it is ensured that only the receiver in possession of the secret key $k_s \in \{0, 1\}^Z$ can read the data. The same concept can now be used to authenticate the sender of a message. Here, a sender uses her or his secret key $k_s \in \{0, 1\}^Z$ to sign a message and create a digital signature. Later, all parties possessing the corresponding public key $k_p \in \{0, 1\}^Z$ can authenticate the sender by verifying the signature since only the private/secret key owner can create a particular signature. In this work, such asymmetric authentication operations are denoted by the function $\sigma(D, k)$, where a public/private key pair is described as $k \in \{k_p, k_s\}$. In this way, the data $D$ can be signed with the secret key $k_s$ and verified by the public key $k_p$. Compared
to symmetric key operations, asymmetric key methods are much more computationally expensive. Therefore, especially lightweight devices often make use of symmetric encryption algorithms rather than asymmetric ones. Today’s cryptographic protocols, however, typically use a combination of symmetric and asymmetric cryptography to securely exchange a symmetric key, as detailed next.

**Diffie-Hellman Asymmetric Key Exchange**  One method to securely exchange symmetric keys over an untrusted network, is the well-known Diffie-Hellman key exchange algorithm [DH76]. Here, two communicating parties are each assigned a public and a private key. By exchanging their public key with each other, both parties can compute a shared secret, e.g., a symmetric AES key, without having to send it over the untrusted network. This work makes use of this method in Chapter 5 to provide a secure channel for transmitting hardware/software configurations to PSoCs deployed in an untrusted environment. In doing so, the Diffie-Hellman key exchange can be implemented using different mathematically difficult to solve problems, as explained next.

**Elliptic Curve Cryptography (ECC)**

The most common public/private key algorithms in use today are either based on the difficulty to factorize two large primes, as in the case of Rivest-Shamir-Adleman (RSA) [RSA78], or take advantage of the fact that no efficient solution exists to solve discrete logarithms, as in the general case of Elliptic Curve Cryptography (ECC) [Kob87]. Both approaches are based on computationally intensive mathematical problems, making their implementation on resource-constrained devices, such as the PSoCs proposed in this work, challenging. To select the right asymmetric scheme according to the given security requirements for a resource-constrained embedded device, both algorithms are compared based on their security strength in relation to their symmetric counterparts in Table 2.1. The table shows that the ECC method requires rather small key sizes compared to the RSA method to achieve the same level of security. In fact, with the RSA method, the required key length literally explodes to accomplish the same computational security. In turn, this results in private key storage requirements being 30 times larger than for a private key in the ECC method. However, this work aims to avoid off-chip key storage – which an attacker could manipulate – by applying so-called PUFs. While the concept of PUFs will be described in detail in Chapter 3, it can be stated that the amount of hardware resources required for PUF-based key generation rises with key length. It follows that ECC algorithms with their small key sizes are better suited not only to be implemented on low-cost CPUs but also for PUF-based key generation on FPGAs. Among this, smaller key sizes lead to faster en-/decryption, lower memory requirements as well as power consumption, and additionally reduces bandwidth requirements, especially for applications that demand higher security levels.

The underlying mathematical foundations, fundamental to the concept of ECC, as needed for the secure update of PSoCs in Chapter 5, are briefly described in the following.
An elliptic curve $E_F$, defined over a finite field $F$, represents the set of points $P = (x, y) \in F$ that solve the equation

$$y^2 = x^3 + ax + b, \quad a, b \in F.$$  

(2.1)

Three arithmetic operations on such curves are fundamental to this work and are described next.

ECC Arithmetic Operations The public-key cryptographic scheme applied in the update protocol requires arithmetic operations over a predefined elliptic curve. In particular, this includes operations on this curve for point addition and doubling required for scalar multiplication. These two operations will be exemplified for a Koblitz secp256k1 curve defined over $y^2 = x^3 + 7$, with $a = 0, b = 7$, as illustrated in Figure 2.6a, respectively, Figure 2.6b. For simplicity, the shown curve is plotted over an infinite set of real numbers $\mathbb{R}$ instead of the specified finite prime field $\mathbb{F}_p$, with $p$ denoting an odd prime number. Figure 2.6a exemplifies point addition, which exploits the following elliptic curve property: a non-vertical line intersecting two non-tangent points, $P$ and $Q$, always intersects with a third point $-R$ on the curve. By mirroring the point $-R$ on the x-axis, one obtains the sum of $P + Q = R$ corresponding to a point addition. Point doubling, as shown in Figure 2.6b, uses a second property that defines that a line tangent and not vertical to the curve at one point, suppose $P$ again, to intersect exactly on one other point $-R$ on the curve. Again, by mirroring the point $-R$ on the x-axis, the result $2 \cdot P = R$ is obtained corresponding to point doubling.
Suppose now a generator point \( G = (x, y) \) denotes a known point on the elliptic curve \( E_F \), and \( d \) denotes a scalar number. In that case, a point \( Q \) can be calculated by the arithmetic operation of scalar point multiplication \( Q = d \cdot G \). Point multiplication on an elliptic curve is analogous to exponentiation in a multiplicative group [Sta09]. For this reason, the inverse operation recovering the scalar value \( d \), when \( G \) and \( Q \) are known, denotes the underlying discrete logarithm problem shown in Eq. (2.2):

\[
d = \log_G(Q)
\]

(2.2)

With today’s computers, this problem is considered to be infeasible to solve\(^2\). Consequently, the secret key \( k \in \{0, 1\}^\mathbb{Z} \) in elliptic curve cryptography is the scalar number \( d \). The public key is the calculated point \( Q \) on the curve, where scalar point multiplication can efficiently be calculated by the so-called “double-and-add” method. This method iteratively performs a combination of point addition and point doubling for point multiplication. An example for \( d \cdot G \), where \( d = 5 \) is shown in Eq. (2.3):

\[
5 \cdot G = 2 \cdot P + 2 \cdot P + P = Q,
\]

(2.3)

which requires in total two point additions and two point-doublings. From this, it can be followed that the execution time for elliptic curve cryptosystems is dominated by the time needed to perform scalar point multiplication for large integers \( d \). Therefore, the calculation of ECC in binary fields, which is more efficient for hardware implementation, is introduced next.

**The Finite Field \( \mathbb{F}_{2^K} \)**

Elliptic curve cryptography is usually defined over two distinct fields \( \mathbb{F} \). One is the prime field \( \mathbb{F}_p \), and the other is the characteristic 2 field of the form \( \mathbb{F}_{2^K} \), often referred to as binary fields. Previous work [RS04; WHII] has shown

<table>
<thead>
<tr>
<th>Security Strength [Bits]</th>
<th>Symmetric Sche.</th>
<th>RSA Key Size(^2) [Bits]</th>
<th>ECC Key Size(^2) [Bits] in ( \mathbb{F}_{2^m} )</th>
<th>ECC to RSA Key Size Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \leq 80 )</td>
<td>2TDEA</td>
<td>1024</td>
<td>163</td>
<td>1:6</td>
</tr>
<tr>
<td>112</td>
<td>3TDEA</td>
<td>2048</td>
<td>233</td>
<td>1:9</td>
</tr>
<tr>
<td>128</td>
<td>AES-128</td>
<td>3072</td>
<td>283</td>
<td>1:12</td>
</tr>
<tr>
<td>192</td>
<td>AES-192</td>
<td>7680</td>
<td>409</td>
<td>1:20</td>
</tr>
<tr>
<td>256</td>
<td>AES-256</td>
<td>15360</td>
<td>571</td>
<td>1:30</td>
</tr>
</tbody>
</table>

\(^2\) Estimates of the security strength of algorithms based on either the prime number factorization problem (RSA) or discrete logarithm problem (ECC) are significantly affected when quantum computing becomes a practical consideration, see also Section 5.4.
that custom hardware implementations benefit from the more efficient computation within binary fields compared to prime fields. Beneficially, modular addition can be performed on characteristic 2 fields without carry-propagation since the polynomial addition required in this process is only an XOR-operation and is, therefore, particularly well suited for hardware implementations. For this reason, the methods proposed in this thesis make use of arithmetic point operations in binary fields with \(2^K\) elements, where the security parameter \(K\) applies for \(K^2=2^{163}\) bits (cf. Table 2.1). Here, the field elements are represented by a set of binary polynomials of degree \(K\) in the following form:

\[
a_{K-1}x^{K-1} + a_{K-2}x^{K-2} + \ldots + a_1x + a_0, \quad a_i \in \{0, 1\}.
\]

(2.4)

According to [Sta09], addition and multiplication are then defined in terms of an irreducible binary polynomial \(f(x)\) of degree \(K\), as follows:

- **Addition:** If \(a = a_{K-1}x^{K-1} + \ldots + a_0\), \(b = b_{K-1}x^{K-1} + \ldots + b_0 \in \mathbb{F}_{2^K}\), then \(a + b = r\) in \(\mathbb{F}_{2^K}\), where \(r = r_{K-1}x^{K-1} + \ldots + r_0\) with \(r_i \equiv a_i + b_i \pmod{2}\).

- **Multiplication:** If \(a = a_{K-1}x^{K-1} + \ldots + a_0\), \(b = b_{K-1}x^{K-1} + \ldots + b_0 \in \mathbb{F}_{2^K}\), then \(a \cdot b = s\) in \(\mathbb{F}_{2^K}\), where \(s = s_{K-1}x^{K-1} + \ldots + s_0\) is the remainder when the polynomial product \(a \cdot b\) is modulo divided by \(f(x)\) and all coefficient arithmetic performed modulo 2, as required for addition.

Here, \(f(x)\) denotes the curve-specific irreducible polynomial to realize these operations for a finite amount of elements by polynomial modulo operation. Finally, the communicating parties must agree on specific curve domain parameters to establish an asymmetric cryptographic scheme, such as digital signatures based on elliptic curves. For a curve defined over the field \(\mathbb{F}_{2^K}\), these are, in addition to the security element \(K\), the irreducible polynomial \(f(x)\), the generator point \(G\), two integer elements \(a, b\), a fixed prime \(n\), and a cofactor \(h\). A given septuple \(\mathcal{D} = (K, f(x), a, b, G, n, h)\) defines, thus, not only the curve on which the communicating parties agreed but also the computational security for this communication. In addition, these parameters determine the processing overhead and storage requirements for the usage of ECC in an FPGA-based edge device, as evaluated in Chapter 5.

### 2.3.3 Cryptographic Hash Functions

Cryptographic hash functions are used in this thesis for basic information security methods, including a) verifying data integrity, b) generating and verifying digital signatures, and c) deriving cryptographic keys by using so-called Key Derivation Functions (KDFs). Here, a cryptographic hash function generally refers to a permutation function that compresses an input of arbitrary length into a fixed-length hash digest [Nat12]. Moreover, these functions are one-way functions that ensure that the computation only works
in one direction, i.e., the original input cannot be derived or back-calculated from the hash value. This one-way characteristic allows checking the integrity of a message transmitted or stored in an untrusted environment by computing a cryptographic digest over the data that the message represents. If even a single bit of the message changes or gets manipulated, the entire hash output is affected and differs. Due to the length of the digest, it is impossible for an attacker to determine the input of the hash function using a known hash. Today’s most common hash function is the so-called Secure Hash Algorithm (SHA)-2, which the NIST in [Nat12] standardized. Operations involving the calculation of a hash are written throughout this thesis as $\mu(D)$, where $\mu$ denotes the computation of the cryptographic hash function over the data elements $D$.

SHA-2 defines output lengths of 512, 383, or 256 bits that can be used for integrity proofs. In the SHA-256 algorithm, used in this work, the input data $D$ is divided or padded always to a multiple of 512 bits. Each block of 512 bits is further divided into 16 words each with a bit width of 32 bits. To compute a hash length of 256 bits, these 16 words must be processed in a sequential loop with 64 iterations, making it suitable for efficiently processing in hardware. A recent alternative to SHA-2 is the SHA-3 algorithm standardized by NIST in 2015 [Nat15], which is based on a fundamentally different block transformation function, namely the Keccak-f[1600] function. This standard includes the two cryptographic hash functions SHA3-256 and SHA3-512 and the two extendable-output functions SHAKE-128 and SHAKE-256, where the appended number indicates the digest length. Note that the SHA-3 hash function standard is not a replacement for the SHA-2 standard due to security concerns but merely an addition to the SHA-2 family with an alternative permutation approach. Therefore, functions based on SHA-2 can alternatively be replaced or upgraded by the aforementioned SHA-3 hash functions. This upgradeability applies to all SHA-2 operations utilized within this work.

Keyed-Hash Message Authentication Codes (HMACs) As introduced in Section 2.3.1, as a form of symmetric cryptography, MACs, combined with a symmetric secret key, are mechanisms for message authentication and integrity checking. In this context, an HMAC operation applies any of the introduced cryptographic hash functions $\mu(D)$ to compute a message digest over a block of input data $D$ in conjunction with a symmetric key $k_s \in \{0, 1\}^Z$ of length $Z$, which will be represented throughout this work as a function of $\eta(D, k_s)$. HMACs, as standardized by the NIST in [Nat08], are primarily used in this work to verify that a sender of a message, respectively, data is trustworthy by concatenating the input message/data with the symmetric key beforehand. Since the sender and receiver have agreed on this key as a shared secret in advance, both parties can authenticate each other. Apart from integrity checks, these secure hash computations offer the advantage of storing data in a compact and secure form, thus, allowing it to be held on non-volatile memory without incurring large storage overheads. Such hash-reduced information is applied, for example, in Section 3.4.2 for securely
providing helper data for key generation, where a list of HMAC generated cipher values, denoted as $C$, are stored instead of plaintext.

**Password-Based Key Derivation Functions (PBKDFs)** For the security of cryptographic applications, the randomness of cryptographic keys is essential. The presented work exploits the properties of so-called PUFs for cryptographic key generation, which practical instantiation will be discussed in detail in Section 3.4. However, due to a non-optimal entropy under some circumstances and possibly not complete randomness of PUFs, a PUF output is not suitable to be used directly as a cryptographic key. Additional measures are necessary to ensure the randomness of cryptographic keys derived from them, as explained in the following.

NIST has standardized a method, called Password-Based Key Derivation Function (PBKDF) [Nat10], for this purpose, where their in- and outputs are illustrated in Figure 2.7. In this method, a cryptographic key $k \in \{0, 1\}^Z$ of length $Z$ is derived from a set of private data (e.g., a password) with insufficient randomness and length, denoted $w$ in the following. Here, non-private data with high entropy and sufficient length $R$ is required to form a salt. Such a salt, denoted throughout this work as $S \in \{0, 1\}^R$, is used as input data to generate cryptographic keys of desired length $Z$ from a given secret input $w$. This operation is referred to as $PBKDF_{\eta,CNT} (w, S, Z)$, where $\eta$ is a standard hash-based message authentication code as permutation function. In addition, a fixed iteration number, denoted as $CNT$, determines how many times this $\eta (D, k_s)$ is applied to permute the key $k \in \{0, 1\}^Z$. The key derived from this procedure is computationally indistinguishable from a truly random output and can, thus, serve as a cryptographic key. Again, a concrete, practical instantiation of PBKDF for secure key generation will be elaborated in Chapter 3.

### 2.3.4 FPGA Key Storage and Device Authentication

In the previous subsections, we have seen that the security of the proposed cryptographic measures is based on the confidentiality and integrity of secret keys. Therefore, this section shall first explain how key provisioning is handled on the current generation of
FPGA devices. Subsequently, two hardware-based methods are discussed that can be used for device authentication, which raises the bar for attackers to copy or steal keys, thus, preventing unauthorized access to private data.

**FPGA Vendor Anti-Tamper Protection** To reduce the risk of key theft and provide measures to ensure bitstream confidentiality and secure device authentication, FPGA vendors have offered on-chip anti-tamper features on their devices for many years [TM14]. On Xilinx FPGAs, for instance, hardwired on-chip crypto modules are available that support AES for symmetric cryptography and RSA for asymmetric cryptography. Permanent key storage for these modules is provided either by the option to store the key within reprogrammable battery-backed RAM (BBRAM) or within eFuses registers which, however, are only one-time programmable. Together with the secure key storage, the crypto modules form the basis for tamper protection measures built on top of them to ensure the confidentiality and integrity of the bitstream by enabling symmetric encryption, respectively, authentication but also asymmetric security techniques. However, these vendor-provided security techniques are limited and insufficient for a comprehensive security-oriented system architecture. For instance, neither the build-in AES nor the RSA module is accessible from the programmable logic and only supports operations in one direction. In particular, this means that the RSA implementation can only verify signatures but cannot generate a signature itself. Both crypto modules are even more limited insofar as they can only be used for the device’s boot process and cannot be configured to encrypt or sign other data. In addition, due to their fixed wiring, these modules cannot be adapted to dynamic threat scenarios; thus, no crypto-agility is possible [WAG+21]. This work identifies solutions that build on existing FPGAs/PSoC security techniques that lead to a deeply layered security to address these issues. For example, both the PSoC boot and update protocol must be able to create and verify asymmetric signatures as well as de- and encrypt in both directions. Hence, the proposed security-oriented system architecture is designed to provide the necessary flexibility to cover all required cryptographic algorithms. To solve the problem of permanent key storage, the approach proposed in this work uses keys generated at runtime within the reconfigurable logic of the FPGA and, thus, are re-provisioned at each system startup.

**FPGA Vendor Device Authentication** Trust can only be established if the communicating parties can identify themselves unambiguously. In this context, hardware-based measures isolated from the application processor represent a more trustworthy security layer since they are usually designed to ensure the security of sensitive data even if the processor should get compromised. For this purpose, FPGA manufacturers program identification numbers into their devices to provide the option to authenticate them later in the field uniquely. In the case of Xilinx devices, this is a unique, read-only 57-bit board-level identifier called Device DNA [TM14]. This device DNA, in the following denoted as ID_{SoC}, is burned into an eFuse register during the manufacturing process and,
if Joint Test Action Group (JTAG) for programming has been disabled, can only be read internally by the FPGA hardware. To subsequently authenticate a PSoC/FPGA device, a custom hardware design can be loaded that reads this register after power-up and grants permission to proceed after successful authentication. Similar methods are also offered by other FPGA vendors such as Intel/Altera via their ALTCHIP_ID Port [Int20]. At first glance, this hardware-based authentication may seem beneficial because the device DNA can only be read internally on the device. While this is true for protection against cloning a hardware design or misuse of a particular PSoC, it does not protect against malicious bitstream modifications or use of the same FPGA with a different bitstream. Another disadvantage is that although these identification numbers can only be read device-internally, they are still not completely secret, as the FPGA manufacturer burns in the keys during the chip’s manufacturing process. Furthermore, it cannot be ruled out that the entire supply chain also has access to these IDs since debug ports such as the JTAG interface – which are enabled by default when chips are shipped – easily allow these identifiers to be read. As can be seen, the DNA on its own is never enough to adequately protect a system. Nevertheless, it can be part of a multi-layered security concept as it will be proposed in Chapter 4 for secure boot.

Vulnerabilities of Secure Key Storage

In general, neither hardwired crypto instances nor cryptographic key storage can provide absolute security guarantees. Indeed, in a potentially insecure environment where an adversary has physical access, the risk of key theft and malicious tampering with such permanent storage is omnipresent. This threat arises from the fact that new attacks are published repeatedly, challenging the security of physical key storage on fielded devices.

In this context, it has been shown that, in particular, the backside of these devices is susceptible to semi-invasive attacks. For example, optical defect analysis methods are used here, which are standard practices in semiconductor manufacturing and testing [RA19]. These inspections involve lighting the backside of the chip with optical methods because, unlike the front side, no special chemicals are required to remove the chip package and etch away metal layers that serve to protect the underlying silicon. For example, in [Sko10], Skorobogatov proposed a semi-invasive attack in 2010 that uses optical fault injection to bypass verification through keys stored on secure embedded flash memory. Another confirmed semi-invasive attack possibility is the readout of eFuses-based key storage with a scanning electron microscope [TM14]. More recently, increased attention has been paid to Thermal Laser Stimulation (TLS), which is primarily used in IC testing to locate metallic short circuit faults in memory cells. For TLS, a laser is used to create a local heat spot on a memory cell that causes thermal effects in the underlying transistors that, assuming a constant voltage supply, result in a change in the current drain depending on their gate state (0 or 1). This current drain is typically in the range of only a few nA but can with an amplifier measured and extracted. By correlating the induced current variations with the laser spot, it is possible to obtain a
response map of the memory cells, revealing values stored in them. The setup of such a TLS measurement is shown schematically in Figure 2.8. An adversary targeting a device’s memory could use this technique to read out cryptographic keys or other sensitive data stored in plaintext. The work in [LTK+18; KLS+20] demonstrated this attack by showing it is possible to extract the 256-bit AES key from the battery-backed RAM (BBRAM) of Xilinx FPGAs even with a commercial and rather cheap laser fault injection station. These attacks show that semi-/non-invasive optical approaches pose a significant threat to the confidentiality of cryptographic keys, as they provide a feasible way to read out persistent memory with a modest investment of time.

The need for secure key provisioning alternatives stems from the risk of key theft through physical attacks, as edge devices encounter due to their deployment in untrusted environments. For this reason, this thesis develops methods, techniques, and protocols based on the security of so-called Physical Unclonable Functions (PUFs) to generate and provide unique FPGA-internal keys for authentication and encryption at runtime. The fundamental security properties of PUFs make them a particularly promising scheme for designing trusted edge devices, as they provide a viable, and above all, secure alternative to permanent key storage in an untrusted environment. According to the well-known Kerckoffs’ principle, an attacker can know all details about the cryptographic methods except for the key itself.

“A design is secure if everything is known about it, except the key”

― Extension Kerckhoffs’ principle, Auguste Kerckhoffs (1883)
For this purpose, the following chapter proposes the concept of CHOICE, a signature-tunable PUF to combat the threat of key theft by providing keys derived from FPGA-intrinsic secrets.
Cryptographic keys are the basis of trust for almost all cryptographic operations, be it authentication, encryption, signing, or key exchange. This makes it a top priority to keep these keys secret under all circumstances. However, as described in Chapter 2, cryptographic keys are often stored in external key storage on the board itself, which seriously compromises this approach by leaving them at risk of malicious tampering in a potentially hostile environment.

For this reason, Physical Unclonable Functions (PUFs) have emerged over the last twenty years as a simple yet tamper-resistant cryptographic primitive to avoid the risks that go along with permanent key storage [CZZ17]. PUFs can be seen as physical entities that provide a defined output, a so-called response, to a specific measurement of physical conditions, from which cryptographic keys can be derived. By taking such a radically different approach for the generation and provisioning of secret keys, PUFs form an undeniable alternative to traditional keys and key management. Their unclonability stems from certain fundamental random physical phenomena that are converted into a digital representation. One place where such random physical phenomena can be observed is in the manufacturing process of today’s nano-scale Integrated Circuits (ICs), producing uncontrollable inaccuracies. Here, so-called silicon PUFs provide a secure way to transform these inaccuracies into a unique and unpredictable “Fingerprint” for each individual IC. As such a fingerprint is by construction physically unclonable it qualifies not only to generate device-specific keys from it but also to track Intellectual Property (IP) [GKS*07a] and uniquely authenticate devices in the field [SD07]. Moreover, invasive and semi-invasive attacks such as the Thermal Laser Stimulation (TLS) introduced in Section 2.3.4 have a negative impact on the behavior of the PUF response, altering its digital representation and rendering it useless. Therefore, such tamper evidence offers a considerable increase of security to protect the three central security goals integrity, confidentiality, and authenticity when a PUF is chosen over traditional key storage on fielded edge devices.

\(^1\)Fingerprint: A physically-defined output that qualifies as a unique identifier.
Nevertheless, specific challenges still have to be addressed for a broader use of silicon PUFs outside a fixed laboratory environment. For instance, on identical devices, it is often difficult to ensure sufficient randomness to generate unique identifiers. In addition, PUFs must be reliable in the sense of a low number of bit deviations across a repetition of PUF response measurements. The biggest challenge, however, is their sensitivity to aging and environmental changes, particularly to changing temperature conditions or voltage supply variations. In this context, it is well known that ambient temperature affects the operation of ICs by changing the junction temperature, and thus, the delay of transistors and wires, which directly affects the PUF response. Similar effects can be observed for long-term IC stress, known as transistor aging. All these effects have the potential to alter the security-related properties of the PUF and, in the worst-case, even render the PUF insecure. To address these challenges, this chapter proposes one of the first ever tunable PUF designs called CHOICE \[\text{SKB}^*+\] for FPGA targets. Here, the statistical properties of the PUF response can be directly controlled by address-tunable propagation delays. As will be shown, this tuning is beneficial to achieve a higher uniqueness of the PUF without sacrificing randomness. In addition, the tuning even allows an initial PUF configuration to be updated to prevent the PUF from becoming insecure over time in the face of continuous aging.

The remainder of this chapter is organized as follows\(^2\). Section 3.1 provides an overview of the security foundations of silicon PUFs and introduces the main factors of process variations within today’s nano-scale transistors. This is followed by a brief introduction of temperature and aging effects and their direct impact on the robustness of the PUF response. The proposed tunable PUF architecture is presented in Section 3.2, and its properties are demonstrated through a series of experimental results in Section 3.3. Finally, Section 3.4 demonstrates the practicability of the PUF within a cryptographic key generation scheme that is used throughout this thesis to make external key storage obsolete. Section 3.5 concludes the chapter with a brief summary.

### 3.1 Silicon Physical Unclonable Functions (PUFs)

A silicon PUF, also called a physical random function, is a circuit that exploits the physical properties of its underlying circuitry and is typically implemented on conventional ICs without requiring additional hardware [GCV*02; AMS*11]. In the context of this work, the silicon realization is limited to FPGA-based PUF instances, specifically delay-based PUFs. Delay-based PUFs exploit variations in timing behavior to convert these into a unique IC fingerprint, also called the signature of the IC. While the timing variations induced by the routing on an FPGA between logic blocks can undoubtedly be manually adjusted, the adjustment of wires within the FPGA’s Configurable Logic Blocks (CLBs), as introduced in Section 2.1.1 and illustrated in Figure 2.3, is very limited. As such, the timing

---

\(^2\)The contents of this chapter are based on and have been partly published in [SKB*21a] and [SKB*21b]. In particular, major parts of Section 3.2 and Section 3.3 are published in [SKB*21b].
variations are derived entirely from process variations as they result from inaccuracies in the IC manufacturing process that are beyond the control of the manufacturer.

The unpredictable and uncontrollable nature of these process variations makes the PUF itself unpredictable and physically unclonable. This characteristic renders it highly unlikely to find two PUF instances that provide the same signature even on identical devices. In practice, however, several factors can negatively affect a PUF’s response, potentially compromising the security properties that PUFs promise. Therefore, this section, first, outlines in Section 3.1.1 the security fundamentals of PUFs before Section 3.1.2 briefly discusses the contributors and effects of process variations on nano-scale transistors in silicon PUFs. After that, Section 3.1.3 further elaborates their sensitivity to temperature variations and transistor aging before Section 3.1.4 provides an overview of related work on silicon and, in particular, delay-based PUFs.

3.1.1 Security Foundations

The security of the PUF proposed in this thesis is based on random timing characteristics of the underlying circuit structure. These timing characteristics can be explored through a PUF measurement denoted as \( m \), which causes a PUF response that denotes as \( r \). Before discussing the specific security properties of CHOICE, the following subsection lays down the general fundamentals and assumptions of silicon PUF security.

The notion of the unpredictability of a random function \( \mathcal{P} : \{0, 1\}^* \rightarrow \{0, 1\}^* \) can be explained by the following security experiment adapted from [AMS+11]. An attacker, let us call her Eve, learns the outputs \( y \) of \( \mathcal{P}(x) \) from a set of inputs \( \{x_0, \ldots, x_{n-1}\} \) during a learning phase. In a second phase, called the challenge phase, Eve has to return her values \( (y, x) \) for some \( x \notin \{x_0, \ldots, x_{n-1}\} \) after a specific time \( t \). The random function \( \mathcal{P}(x) \) is unpredictable if and only if Eve fails to predict the output \( y \) of \( \mathcal{P}(x) \) for inputs \( x \notin \{x_0, \ldots, x_{n-1}\} \) with a probability at least \( 1/2 \).

Such a security experiment defining unpredictability can be applied to the security requirement of an unpredictable PUF response. Given the irregular timing characteristics of a specific circuit, it is assumed that an attacker cannot predict a specific response for a given device from a set of previously observed responses. In other words, the PUF signature of a given device is indistinguishable from random. Moreover, it is assumed that due to the inherent properties of the PUF, a malicious party with physical access, e.g., performing an invasive attack (cf. Section 2.2 and Section 2.3.4), further contributes to timing irregularities, and thus, negatively affect the PUF response. Rührmair et al. [RSS09] described such PUFs as so-called obfuscating \( t \)-PUF (also stated as Weak PUF) by the following notation. Let \( \mathcal{P}(m) \) be the measuring of a physical function with inherent randomness properties on a physical system \( b \), i.e., in the context of this work, an FPGA. Further, \( b^{\mathcal{P}(m)} \) denotes the measurement result that is obtained on a given measurement \( m \). Then according to Rührmair et al. [RSS09], \( \mathcal{P} \) is called an obfuscating \( t \)-PUF when

\[
b^{\mathcal{P}(m)} = r^b_{\mathcal{P}(m)}
\] (3.1)
Cryptographic Key Provisioning using Physical Unclonable Functions

hold. It is essential that the response \( r_{b}^{(m)} \) of a silicon PUF results from uncontrollable random process variations, at least in parts. That means it should be practically infeasible for an original manufacturer of \( b \) to produce another system where a measurement would result in the same response. Consequently, the PUF outputs measured should be unique and independent even on quite similar physical systems. As a result, the above security experiment for a PUF can be reformulated as follows.

In the learning phase, a single PUF measurement \( m \) on a number \( N \) of different devices leads to \( r_{b_0}, \ldots, r_{b_{N-1}} \) individual responses. During the challenge phase, Eve must then predict the output \( r_b \) for another device \( b \), where \( b \not\in \{b_0, \ldots, b_{N-1}\} \). If Eve fails to predict \( r_b \) after a time \( t \), it can be stated that the PUF provides enough randomness even on quite similar devices. Apparently, there is a fundamental difference between the general definition of unpredictability and this PUF experiment [AMS+12]. Eve evaluated exactly one random function in the previous notation, but many challenges. In the second experiment, many different random functions must be learned, but only for one fixed challenge at a time. This is due to the entanglement of the PUF and the measurement apparatus, which ultimately defines the PUF response.

It can be concluded that even if an attacker knows everything about both the measurement apparatus \( b \) and the PUF, he or she can only predict the response \( r_b \) if at least one measurement \( m \) is conducted with the PUF on the exact device \( b \). However, even in this case, one measurement is not necessarily enough to accurately predict every bit of the PUF signature due to measurement noise, which defines a PUF’s reliability. Before discussing the term reliability in more detail, the concept of the nominal response of a PUF is defined as the most frequently occurring response in a number of \( M \) independent measurements. In addition, this most occurring response is denoted as \( r^* \), which allows to define the indicator of uniqueness in the following.

**Uniqueness** The uniqueness of a given silicon PUF is defined by the average inter-die uniqueness between devices. Hereafter just called uniqueness. The uniqueness of a PUF response is calculated by comparing the difference in bits of the nominal response obtained from \( M \) measurements on a single device against the nominal responses of a number \( N \) of other devices. This difference comparison is performed based on the Hamming distance \( HD \) of the responses \( r^* \in \{0, 1\}^Z \), where \( Z \) denotes the length of the response. Therefore, the equation of uniqueness is calculated as:

\[
\text{Uniqueness} = \frac{2}{N \cdot (N-1)} \cdot \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} \frac{HD(r^*_i, r^*_j)}{Z} \cdot 100\%,
\]

where \( r^*_i \) is the nominal value of a device instance \( i \). Here, a Hamming distance close to \( Z/2 \) for each pair of devices is desirable. Such a distance would result in a PUF uniqueness of 50% across all devices, which is considered to be optimal. The unpredictability as described then arises because the devices differ, on average, in half the number of their
three.prop/period.up/one.propup Silicon Physical Unclonable Functions /parenleft.upPUFs/parenright.up bits. A uniqueness that exceeds the 50% and tends towards 100% again degrades the security properties of a PUF, as the PUF response becomes predictable.

A brief example to illustrate this: Let \( r \) be a PUF response with \( Z = 4 \)-bit in size and let the PUF’s uniqueness be 100%. Let us further assume that this PUF provides on a given device \( b_0 \) the response \( r_0 = 0110 \). Due to the PUF’s uniqueness properties, the PUF’s response becomes predictable for a second device \( b_1 \), as a measurement on a second device would reveal either the same response or precisely the inverse, namely \( r_1 = 1001 \). If the PUF has a uniqueness of 50%, six possible responses for the device \( b_1 \) need to be considered: 1100, 0011, 0101, 1010, 1111, and 0000, which corresponds to a binomial coefficient of \( \binom{4}{2} = 6 \). For the uniqueness of 75%, it would be \( \binom{4}{3} = 4 \), and for 25% uniqueness, it would be \( \binom{4}{1} = 4 \). This illustrates that a uniqueness close to 50% is optimal, and thus, desirable.

For real-world PUF implementations, a response length of \( Z = 128 \)-bit or larger is typically chosen, which would lead e.g., for a response length of \( Z = 128 \)-bit and 50% uniqueness in \( \binom{128}{64} = 2.4 \cdot 10^{37} \) possible permutations. This results in the fact that predicting the response on any other board is close to random guessing for a PUF response of 128-bit in length and a proven uniqueness close to 50%. Moreover, as stated above, a fixed static secret does not exist as every measurement contains additional uncertainty. This uncertainty is also defined as the reliability of the PUF as detailed in the following.

Reliability Besides uniqueness, reliability is another critical property of silicon PUFs. This property characterizes the variability of a random PUF function \( \mathcal{P}(m) \) (cf. Eq. (3.1)) in dependence of different measurements, as these typically contain a certain amount of bistable information that can vary even under constant measurement conditions. Such bistable bits manifest themselves in measurement noise, making it a source of dynamic entropy in the response \( r_b^{\mathcal{P}(m)} \) of a silicon PUF. However, if this noise is too high, excessive measurement errors will occur, degrading the reliability of the PUF and, in the worst case, compromising its security. A PUF’s reliability is defined by the Bit Error Rate (BER) on a device \( b \) and calculated as the average Hamming distance \( \text{HD} \) over all \( M \) measurements from the nominal value \( r_b^* \in \{0, 1\}^Z \) with length \( Z \) in bits, which can be expressed as follows:

\[
\text{BER}_b = \frac{1}{M} \cdot \sum_{m=0}^{M-1} \frac{\text{HD}(r_b^*, r_b^m)}{Z} \cdot 100\%.
\] (3.3)

As each response measurement provides only a snapshot of the physical properties of a PUF under certain environmental conditions, \( r_b^{\mathcal{P}(m)} \) is simply used as \( r_b^{\mathcal{P}(m)} \) in the remaining work.

From Eq. (3.3) it can be seen that an optimal reliability corresponds to a BER of 0%. This reliability is necessary because many cryptographic techniques, such as the
symmetric and asymmetric cryptography presented in Section 2.3.1 and Section 2.3.2, respectively, require a single static secret. However, from an information-theoretic point of view, it should be noted that more robust responses also have lower (dynamic) entropy, and thus, less randomness in the bits. In other words, a BER of 0% is achieved if each PUF bit tends to zero or one with 100% certainty. This is known as the reliability-randomness trade-off of PUFs, where static entropy between devices is desirable. Still, dynamic entropy between repeated measurements can render the PUF insecure or even unusable. Although sophisticated error-correcting codes can be used to correct a certain amount of deviations in an unstable PUF response, it introduces additional latency and resource overheads that grow exponentially with the number of errors to correct. The concrete cost of correctability for a CHOICE PUF in terms of latency and hardware resources is discussed in detail in Section 3.4. To better understand where entropy comes from and how it is affected by changing environmental conditions, the following subsection briefly introduces the significant factors contributing to process variations in silicon PUFs.

3.1.2 Process and Timing Variations

Tiny differences, especially in the timing behavior of identically manufactured silicon circuits, stem from varying propagation delays and timing paths caused by irregular transistor switching characteristics and wire lengths. Random physical phenomena, known as process variations, which are entirely beyond the control of the IC manufacturer, induce these irregularities in timing and result in an inherent uniqueness of each manufactured IC. Therefore, these process variations are the immutable root of the non-clonability properties of PUF circuits, so to say by construction. These properties make copying the circuit useless because another device will always have slightly different timing characteristics. Consequently, copying and rebuilding the PUF circuit on such a device has the effect of creating a new physical function with a correspondingly different PUF response, and therefore is physically unclonable.

The timing behavior of transistor switching as well as their signal propagation time as exploited by delay-based PUFs on FPGAs, is mainly determined by three parameters, namely process variations (P), junction temperature (T), and supply voltage (V). These parameters are referred to as the PVT parameters and corners, central to the Static Timing Analysis (STA) of modern ICs [DH06]. While junction temperature and supply voltage are external factors that are considered controllable to some extend, uncontrollable process variations form the PUF’s inevitable source of entropy. In this context, IC manufacturers generally attempt to reduce the imperfection of the manufacturing process to lower the impact of process variations. However, this is countered by the progressively-shrinking IC structures, which make this attempt more difficult, if not impossible, as small transistor structures tend to lead to even more significant variations.

Today’s Complementary Metal-Oxide Semiconductor (CMOS) technology that combines one p-type and one n-type Metal-Oxide-Semiconductor Field-Effect Transistor
(MOSFET) is the de facto standard fabrication process for transistors [Iwa15], and as such also the foundation of every modern FPGA. Within the FPGAs structure, these transistors are, on the one hand, combined to form corresponding logic gates, thereby allowing the implementation of complex hardware circuits. On the other hand, the transistors are used as programmable switches to route signals, for example, between multiple CLBs on the FPGA. In both cases, process variations can impact signal propagation time, as discussed next.

**FPGA’s MOSFET Process Technology**

With the current generation of nano-scale transistors, several factors can lead to measurable process variations. For instance, the Xilinx Artix®-7-based programmable logic, used in this work, is based on TSMC’s 3 28 nm High Performance, Low power (HPL) process technology utilizing High-K Metal Gate (HKMG) Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFETs) [Xil15]. A schematic representation of such an n-type MOSFET is illustrated in Figure 3.1. In contrast to a standard transistor, the gate oxide is made from a material with a high dielectric constant (indicated by the central dotted area in Figure 3.1). In addition, a metal layer is mounted on the gate oxide instead of the conventional silicon gate layer. The length of the gate denoted as \( L_G \), amounts on this HPL process technology to the actual size of \( L_G = 24 \text{ nm} \) (28 nm refers just to the process name). In contrast to standard transistors, HKMG transistors provide the advantage to offer better performance and lower static power consumption, making them especially suitable for mid-range FPGA devices [Xil15].

However, such advanced CMOS technologies already face today the challenge that their underlying, identically fabricated MOSFETs reveal measurable dissimilarities in their switching performance. Switching speed and, thus, the signal propagation time of a transistor is determined in particular by the MOSFET’s threshold voltage \( V_{th}(T) \) between gate and source, which also depends on the temperature \( T \). Nevertheless, the manufacturing-induced process variations have a considerable influence on this threshold voltage. To ensure that a chip will function correctly even in the presence of process variations, sophisticated timing analysis and modeling techniques are typically used in the IC design process. These timing analysis techniques likewise help to describe the effects that enable and threaten the fundamental properties of delay-based silicon PUFs. Therefore, a short description of the connection between signal propagation delay and threshold voltage for silicon PUFs is provided in the following using the well-established alpha-power law [SN90] of transistor timing.

---

3Taiwan Semiconductor Manufacturing Company (TSMC) is an independent contract manufacturing semiconductor foundry.
MOSFET Timing Characteristics

In the timing analysis of transistors, the propagation time of a signal transition from 0 to 1 or vice versa, e.g., within a CMOS inverter circuit, is modeled by the transistor’s resistance, denoted as $R$, and the internal parasitic load capacitance denoted as $C_L$. This allows the propagation delay to be described as an $RC_L$ time-constant, according to the simplified delay model of the alpha-power law, proposing the delay to be modeled as $\tau \propto RC_L$ [SN90]. For a MOSFET as shown in Figure 3.1, the resistance $R_{DS}$ within the channel between drain and source determines the switching behavior of the transistor. This means that the resistance $R_{DS}$ is high when the MOSFET is “closed”, resulting in “almost” no current flowing between the channel. Closed here refers to the gate-to-source voltage $V_{GS}$ being less than the threshold voltage, i.e., $V_{GS} < V_{th}(T)$. When the gate-to-source voltage exceeds the threshold voltage, i.e., $V_{GS} > V_{th}(T)$ as illustrated in Figure 3.1, drain and source of the MOSFET get connected and form a channel (solid green area in Figure 3.1) with the resistance of $R_{DS}$. The following formula expresses this drain-to-source resistance:

$$R_{DS} \propto \frac{1}{\tau(T) [V_{GS} - V_{th}(T)]} \quad (3.4)$$

\[\text{In practice, effects such as subthreshold conduction and leakage current prevent an absolutely closed state.}\]
Here, $\kappa$ denotes the capacitance factor representing the channel parameters as a function of temperature $T$. On high-k MOSFETs, these channel parameters are defined as $\kappa(T) = \psi_{n/p}(T) C_{ox} \frac{W}{L_G}$, where $\psi_{n/p}$ denotes the charge-carrier effective mobility in the electric field for either an n-type or p-type transistor at a certain temperature $T$, $C_{ox}$ the gate oxide capacitance between the metal layer and channel, and finally $W_G$ and $L_G$ the width and length of the gate. Assuming that $V_{GS}$ is fixed, Eq. (3.4) shows that decreasing the threshold voltage $V_{th}(T)$ leads, according to the alpha-power law, to a decrease in the signal propagation delay $\tau$, since the voltage is inversely proportional to $R_{DS}$. As can be further seen from Eq. (3.4), scaling the gate dimension ratio $\frac{W}{L_G}$ leads to a decrease in the capacitance factor $\kappa$, which also leads to an increase in the transistor’s switching speed and consequently the entire circuit.

However, scaling the transistor dimensions is often accompanied by reducing chip and transistor voltage to reduce its dynamic power. For this reason, the ratio of supply to threshold voltage has steadily decreased over the years with each new generation of technology. Nevertheless, due to static power limitations caused by leakage currents below the threshold voltage, the $V_{th}(T)$ scales much slower than the supply voltage. This divergence enhances the impact of process variations and temperature on the threshold voltage, and thus consequently the transistor switching behavior, which directly affects the PUF’s uniqueness and reliability properties.

Now that the fundamentals of general transistor timing behavior have been covered, the following paragraphs briefly introduce the most dominant contributors to process variations in MOSFETs, since these ultimately affect the properties of silicon PUFs on FPGAs. This short overview is based on the concise summary by Samar K. Saha [Sah10].

**Random Discrete Doping (RDD)** During the manufacturing process of MOSFETs, dopants are induced into the silicon body of the transistor (cf. p-doped silicon body in Figure 3.1) to control the threshold voltage $V_{th}$. However, due to shrinking transistor sizes, the dopant concentration in the transistor channel also decreases, resulting in discernible variations in transistor performance. These process variations are called Random Discrete Doping (RDD) because they result from the randomly scattered discrete amount of dopant atoms in the transistor channel. It is evident that when the number of dopants in the channel is small, an additional or missing dopant atom directly impact the threshold voltage of the MOSFET. Furthermore, the number of dopant atoms is subject to a random distribution. $V_{th}$ also depends strongly on this random distribution, which causes different switching performances between identical MOSFETs. These measurable differences in the electrical switching characteristics between two identical transistors charges are, therefore, known as device-to-device (D2D) performance variability [Sah10], which can be explored by delay-based silicon PUFs.

**Line Roughness** A second primary source of process variations stem from the use of subwavelength lithography during the transistor manufacturing process. When the
transistors to be fabricated are too small to be realized accurately, inaccuracies occur, resulting in imperfect transistor properties. Subwavelength lithography, which is used to pattern transistors, applies a specific wavelength of light, e.g., 193 nm for the 28 nm transistor technology shown in Figure 3.1 and used by Xilinx FPGAs. However, from these numbers, it appears that the wavelength of light used to pattern the transistor is much larger than the transistor’s feature size. As a result, the lithographic process cannot accurately shape the transistor but instead introduces a roughness at the edges and width of the transistor called line-edge roughness (LER), respectively, line-width roughness (LWR). This roughness causes an inconsistency in the length $L_G$ and width $W_G$ of the transistor’s gate, which not only directly affects the capacitive factor $\alpha$ (cf. Eq. (3.4)), but also leads to a $V_{th}$ degradation. The effect of line-edge roughness on the transistor characteristics increases as the difference between the lithography wavelength and the transistor feature size increases. It applies, the more significant the difference, the greater the effect on transistor switching behavior.

**High-k dielectric morphology** Xilinx uses high-k metal gate transistors for its Artix®-7 FPGA series instead of leveraging conventional transistors with gate oxide materials [Xil15]. The main advantage of using high-k dielectrics is that a thicker insulating gate oxide layer can be used as compared to conventional oxide materials, shown by the dotted area in Figure 3.1. As a result, the leakage current of the transistor between drain and source is reduced, thus, decreasing the static power of the entire IC. However, the surface between the high-k dielectric layer and the overlying metal layer is not ideal, as it always reveals some tiny interface roughness. The same is true for the interface between the high-k layer and the underlying silicon layer. These imperfect interfaces cause mobility degradations and gate-oxide variations, which affect the slope of the transistor. In addition, due to the polycrystalline structure of high-k materials, phase separation occurs due to a local crystallization, which in turn leads to variations in the channel conductance.

All of the mentioned sources of process variations are uncontrollable, differ from transistor to transistor, and are, therefore, physically unclonable. The delay-based PUF proposed in this chapter explores all of these variations to create a unique FPGA-intrinsic secret. However, it is important to consider that signal propagation delays are affected also by operating conditions such as temperature and aging effects as well, as both operating temperature and long-term stress affect the internal physical properties of transistors. In the case of temperature, this might be just a short-term effect that may only temporarily impact a PUF measurement. Aging effects, on the other hand, are often irreversible and therefore have the potential to render a PUF insecure. For this reason, it can be inferred that the PUF output depends not only on process variations but equally on the operating temperature and the current device condition. Hence, the following subsection discusses the impact of temperature fluctuations and aging of delay-based PUFs for FPGAs.
3.1.3 Temperature and Aging Effects

As mentioned in Section 3.1.1, a PUF response is not constant for each measurement but varies within a certain error range, affecting its reliability and uniqueness equally. In addition to this dynamic noise inherent in any PUF response, other external factors can potentially degrade the timing characteristics of a delay-based silicon PUF beyond a tolerable error range. In particular, the main factors are thermal noise, supply voltage fluctuations, and long-term stress, which all have the potential to contribute to PUF response errors. Whereas most FPGA applications remain functional within a specific range of temperature and expected lifetime, PUFs are very sensitive to changes in the operating conditions. Consequently, this hampers the reliable reproduction of the signature. For example, a PUF signature may alter over time due to aging or changing operating conditions, preventing the PUF from being used, e.g., in cryptographic key generation. It can be concluded that an initially validated PUF response should be a) robust against temporal variations to ensure its reproducibility even under large temperature fluctuations, and b) updatable to address permanent errors that become noticeable when an initially verified signature deviates over time. Therefore, an overview of the effects of temperature and aging on FPGA-based silicon PUFs is given first before demonstrating in Section 3.3 how CHOICE fulfills the security requirements of an initially validated nominal response.

Temperature Variations

The ambient temperature, hereafter denoted as $T_A$, can vary significantly depending on the application and environment a device is deployed. It is well known that the resistance of a metal wire increases with an increase in temperature, the effect of temperature variations on the propagation delay of nano-scale transistors is not so straightforward. In MOSFETs, temperature affects several transistor characteristics at once, altering the integrated circuit performance. For instance, according to [DH/zero.prop/six.prop] the impact of temperature on the value of the threshold voltage $V_{th}(T)$ as well as of the charge-carrier mobility $\psi_{n/p}(T)$ can be described as the following:

$$V_{th}(T_A) = V_{th}(300) - \theta(T_A - 300)$$

and

$$\psi_{n/p}(T_A) = \psi_{n/p}(300) \left( \frac{300}{T_A} \right)^\chi,$$

where 300 marks the reference ambient temperature in Kelvin, here referring to a room temperature at 26.85 °C, while $\theta$ and $\chi$ denote small positive constants. From Eq. (3.4) in Section 3.1.2, it can be seen that such described temperature dependency directly impacts the propagation delay $\tau$ of the MOSFET transistor since it reveals a temperature dependence in $R_{DS}$. However, according to [DH06], a competition between mobility and threshold voltage can be stated, as a lower mobility decreases the drain current, while...
a lower threshold voltage increases the drain current. It is a competition since both factors determine $R_{DS}$, cf. Eq. (3.4). The final drain current is determined by which trend dominates at a given voltage and temperature pair. Kumar et al. have shown in [KK06] that, in contrast to older process nodes, this makes it difficult to predict transistor speed in sub-90 nm nodes reliably. While the charge-carrier mobility is the primary factor determining drain current at a high gate-to-source voltage, the threshold voltage is critical to drain current at a lower $V_{GS}$ voltage. This is why at a high gate-to-source voltage the device delay increases with temperature, while at a lower gate-to-source voltage the device delay decreases with temperature. This phenomenon, called Inverse Temperature Dependence (ITD), is responsible for the increase in drain current when the temperature is increased, while the gate-to-source voltage $V_{GS}$ remains constant. Since the scaling of the threshold voltage $V_{th}$ lags behind the gate-to-source voltage $V_{GS}$ (cf. Section 3.1.2), temperature-dependent propagation delay characteristics can be expected to have an increased impact on PUF response reliability, as these are constructed from these nano-scale transistors.

Variations in the environmental temperature directly affect the internal junction temperature of the IC, hereafter referred to as $T_J$. The junction temperature refers to the internal temperature of the IC used in the device and is conceptionally interchangeable with the semiconductor’s operating temperature. For Xilinx FPGAs, the supported junction temperature depends on the given device grade, e.g., commercial, industrial, or military. Here, commercial supports the smallest operating temperature range and military the most extensive operating temperature range. In addition, the junction temperature $T_J$ depends not only on the ambient temperature $T_A$ but also on other factors, such as the cooling and power supply of the device and its power dissipation. Consequently, for the commercial-grade Artix®-7 FPGAs used in this work, the ambient temperature $T_A$ should not cause the junction temperature $T_J$ to drift outside the specified operational conditions of 0 °C to 85 °C [Xil20]. In this context, the concept of robustness is defined in Section 3.3.4 as an indicator of the resilience of the CHOICE PUF against extreme temperature variations. It can be shown that CHOICE can guarantee this temperature resilience within the entire operating range of such commercial-grade FPGAs.

High-temperature fluctuations are rather transient effects, but among other factors, lead to device stress, and thus, to continuous aging of the device. For a silicon PUF, this means that measurement errors can manifest themselves, negatively affecting the PUF response in the long term. For this reason, the following paragraph presents the dominating contributors of device aging and discusses their impact on a reliable PUF response generation.

**Aging** Shrinking transistor feature sizes lead to higher electric field strengths and higher current densities, which both accelerate aging and, thus, change important transistor parameters over time [AvSE+14]. In this context, aging refers in particular to...
the reduction of MOSFET switching performance. Here, two effects dominate, firstly the so-called Hot-Carrier Injection (HCI) effect [BGH+09] and secondly the effect of Bias-Temperature Instability (NBTI and PBTI) [WYB+09]. HCI is an irreversible aging effect in which charge carriers with high energy hit the gate insulator layer and remain trapped there. This damages the insulator layer, leading to a change in fundamental transistor properties. In particular, constant electrical stress, whether due to a high transistor switching frequency or a high supply voltage, intensifies this effect and leads to a deterioration of the switching speed, which can in the worst case lead to timing errors.

N/PBTI stress, the second aging effect, is caused by the electric field across the gate insulator, creating dangling bonds at the channel/insulator interface. Particularly in High-K Metal Gate (HKMG) transistors which are the technology used in the FPGAs applied in this work (cf. Section 3.1.2), sustained PBTI stress can lead to a permanent shift in the transistor threshold voltage, often denoted as \( \Delta V_{th} \) [KN13]. Consequently, a higher threshold voltage hampers the switching of the transistor, see also Eq. (3.4).

In summary, aging effects, especially for delay-based PUFs, must be carefully monitored to ensure the security of the PUF throughout its lifetime. Otherwise, a drift in the PUF response may occur over time due to such aging degradation, resulting in the loss of the PUF’s unpredictability properties [MMSII].

In this respect, the CHOICE concept as proposed in this work has the obvious advantage over static PUF designs that a new PUF signature can be created by re-tuning the same PUF circuit with a new configuration to obtain again a reliable and secure PUF response, e.g., when a long-term variation of \( x\% \) (here, \( x \) can be the maximum tolerable error) in the response is observed. The most established silicon PUFs on FPGAs are presented in the following subsection.

3.1.4 Related Work

Before presenting the details of the CHOICE architecture in detail, this subsection will first provide an overview of the most extensively studied silicon PUF approaches on FPGAs. This includes but is not limited to PUFs that exploit glitches in order to be able to assess similarities and differences to CHOICE.

Silicon PUFs on FPGAs

Among the most extensively studied approaches for silicon PUFs on FPGAs are Ring Oscillator (RO) [HWG+18; HMB+19], arbiter [HYK+10; MYI+15], and butterfly [GKS+07a; KGM+08] PUFs. All of these PUF circuits have in common that they are based on comparing delay differences between certain timing paths within the FPGA structure. In this context, earlier work [MKD10; MKP09] has shown that in particular, asymmetries in manual routing between the FPGA’s CLBs can lead to a distortion of these delay differences, reducing entropy, and thus, causing potentially predictable PUF responses.
It follows that strictly symmetric routings are required to achieve equal timing paths and prevent bias, so delay differences stem entirely from uncontrollable process variations, as already outlined in Section 3.1.2. In this context, butterfly PUFs are considered a good solution as they allow to keep both distance and number of freely routed stages to a minimum by cross-coupling latches or simple NOR gates. Arbiter PUFs, on the other hand, as illustrated in Figure 3.2a, require long delay paths wired across multiple multiplexer stages, which inevitably makes them susceptible to bias by asymmetric timing paths [MMS09]. Within the arbiter PUF, a $O$-bit challenge is applied to select individual paths by controlling the multiplexers to compute a single-bit PUF response. The idea is to derive a unique response for each challenge, depending on the relative delay differences between the resulting paths. At the path’s end, a latch representing the arbiter decides which path delay is shorter, corresponding to a PUF bit of either 0 or 1.

(a) Schematic of $O$-stage arbiter silicon PUF in which different challenges $c$ yield to different path combinations. An arbiter latch determines which of the two delay paths is faster, resulting in a 1-bit PUF response (figure based on [HYK’10]).

(b) Schematic of a three-inverter-stage Ring Oscillator (RO) silicon PUF, forming a delay loop that oscillates with a particular frequency, which is measurable from the toggle flip-flop on the right. Additional frequency comparison between two of these circuits – not shown in the figure – is required to generate a 1-bit PUF response (figure based on [HWG’18]).

Figure 3.2: Arbiter (a) and Ring Oscillator (RO) (b) PUFs are among the most established silicon PUFs on FPGAs. Both circuits explore delay differences attributable to process variations. Once a PUF setting is found, no changes can be made, making these PUFs susceptible to changing environmental conditions and aging effects.

Compared to arbiter PUFs, PUFs based on ROs, such as those shown in Figure 3.2b, promise the advantage of not requiring symmetric routings. Here, a RO PUF compares the frequencies of two identical delay loops and generates a single-bit response based on a
simple frequency comparison. However, these PUFs have the disadvantage of high power consumption due to fast switching frequencies and long measurement times to improve the required frequency resolution [HWG+18]. A comparison of the individual security properties of the PUF architectures mentioned above and their resource requirements in terms of FPGA slices per PUF bit follows in Section 3.3 and Table 3.1, respectively.

To conclude, all of these delay-based PUF circuits have in common that once a setting has been found that meets certain uniqueness and reliability requirements, they are considered immutable. Their static design makes them particularly susceptible to the inevitable aging processes described in Section 3.1.3 that cause irreversible changes in transistor switching delays. After all, their inflexibility prevents necessary adjustments to avoid the PUF becoming insecure or even unusable when aging effects manifest.

The concept of CHOICE is comparable to that of arbiter, butterfly, and RO PUFs because it is also based on the principle of delay differences. However, unlike the aforementioned PUFs, CHOICE exploits the effect of glitch generation with timing differences resulting solely from inevitable process variations within hard-wired FPGA elements to enable a new class of delay-tunable and aging-resistant PUF designs.

**Glitch-Based PUFs**

The proposed CHOICE PUF converts individual delay characteristics within the FPGA silicon into a unique device signature. As it will be explained in the following, this is done by detecting the presence of a glitch with the asynchronous preset input of a flip-flop available in the CLBs of the FPGA logic. Here, the internal state of this flip-flop, either 1 or 0, depends on whether the glitch reaches the flip-flop or not, thus, representing a single PUF bit.

The circuit to generate the glitch is inspired by the work of Anderson in [And10]. This circuit, together with the glitch generation, can be seen in Figure 3.3. Here, instead of exploring race conditions within symmetric routings as utilized by arbiter PUFs (cf. Figure 3.2a), a pair of shift registers $S_0$ and $S_1$ as well as a pair of carry-chain multiplexers $MUX_0$ and $MUX_1$ are used to generate the PUF response. Anderson’s method requires that the select inputs of the two multiplexers switch alternately within a common clock cycle to create the glitch, i.e., when the upper multiplexer $MUX_0$ switches from 0 to 1, $MUX_1$ switches from 1 to 0. The alternating sequence used to change the multiplexers is initially loaded via the data input $d$ into the shift registers, depicted on the left side of Figure 3.3. After initializing the shift registers, their outputs $Q$ toggle the connected shift registers upon a PUF measurement. With constant inputs at the lower multiplexer $MUX_1$ (0, resp. 1), a glitch with a certain width, hereafter referred to as $\delta$, occurs at the output of the upper multiplexer $MUX_0$. If $MUX_0$ switches slightly faster than $MUX_1$ – assuming the switching occurs not exactly simultaneously due to the process variations of the internal transistors – a glitch is raised on the wire connected to the preset input of the flip-flop (PRE). A detailed description of the switching behavior of a
Figure 3.3: Glitch-based PUF proposed by Anderson in [And10], consisting of two shift registers connected to carry chain multiplexers (MUX) and a single flip-flop to generate one PUF bit (figure based on [And10]).

CHOICE PUF, together with the corresponding requirements for the signal propagation times, is provided in Section 3.2.2 and Section 3.2.3.

A significant problem with the proposed Anderson PUF is that the design cannot be implemented on the current FPGA generation because the CLB structure in these devices has been redesigned by Xilinx [ZWD+16]. Zhang et al. were the first to describe the problem in [ZWD+16], and at the same time proposed a solution by distributing not only the shift register/multiplexer pairs among the CLBs but also the shift register/multiplexer pair itself. Although this solution allows the Anderson method to be implemented on the current FPGA generation, it has two major drawbacks. The obvious one is that the resource consumption for a single PUF bit doubles compared to the original approach because a total of four slices are now required for glitch generation. The second drawback is the problem of possible bias due to asymmetries in manual routing, as discussed earlier in this section. These routing imbalances can occur because the shift registers are no longer directly connected to the corresponding multiplexers, and therefore, must be manually routed between slices. For this reason, CHOICE takes particular care to ensure that those delay imbalances are only caused by the hard-wired transistors and routings of the available shift registers and carry chain multiplexer primitives within the FPGA.

Usmani et al. described another alternative approach in [UKM+18] to implement the Anderson method on the recent FPGA generation and reduce resource consumption. The authors proposed to replace the shift registers with synchronous toggling signals generated by a flip-flop and a lookup table configured as an inverter. Since this setup
does not rely on shift registers, the carry chain multiplexer can be placed in the same slice as the circuit that generates the toggling signals for the multiplexers to produce the glitch with two slices per PUF bit. In this regard, Usmani et al. were the first to suggest controlling the glitch width $\delta$ to allow a device-specific configuration. It was proposed to route the output of the upper multiplexer through additional multiplexers on the carry chain. These additional multiplexers were declared as filter stages, which allowed the authors to analyze two disjoint PUF configurations with different response characteristics. The CHOICE PUF presented in this work follows the idea of a device-specific PUF setup to achieve better reliability and uniqueness properties by already taking into account the individual conditions of the device. However, CHOICE extends this approach radically by offering a total of 6,144 different configurations to achieve a fine-grained tuning to control propagation delays while keeping resource requirements to a minimum, as it will be explained next.

3.2 CHOICE: A Signature-Tunable PUF for FPGAs

This section describes CHOICE [SKB+21a], a signature-tunable PUF design, as an adaptable alternative to static PUF circuits5. The approach is based on the idea of freely choosing between different delay paths within hard-wired logic elements of the FPGA’s reconfigurable structure. Since the design can be implemented in a single Xilinx slice of type SLICEM, it requires only a minimal amount of FPGA resources. In addition, the PUF’s tuning capability enables device-specific PUF configurations. This tunability makes CHOICE the first PUF to offer the ability to compensate for aging and other long-term stress effects. What makes this possible is the adjustment of the length of the internal circuit timing paths, which directly impacts the signal delays of the PUF, and thus, the statistical properties of the PUF response. As will be shown in Section 3.3, this tuning is beneficial to achieve better uniqueness and reliability properties of a PUF response without sacrificing randomness. Prior to this, Section 3.2.1 introduces the concept of addressable delay adaptation before presenting the CHOICE PUF circuit and its configuration in Section 3.2.2. This is followed by elaborating a delay model in Section 3.2.3 that exemplifies under which timing conditions a glitch occurs in the PUF circuit.

3.2.1 Delay Adaptation

The original circuit for generating a glitch, as proposed by Anderson in [And10], requires two fixed-size 16-bit shift registers. This static design is suboptimal because there is no guarantee that the circuit provoking the glitch will result in a glitch wide enough to generate the storage of a 1 in the flip-flop corresponding to the PUF bit. Due to the length of the routing path in the Anderson design between the upper multiplexer and

---

5Major parts of this section have been published in [SKB+21a; SKB+21b]
the flip-flop (cf. Figure 3.3), which acts as a low-pass filter, the glitch could be filtered out. Indeed, the glitch must have a certain width, denoted $\delta_{\text{min}}$ in the following, to set the flip-flop’s preset input. From this, it can be deduced that adjusting the glitch width $\delta$ increases or decreases the chance of retrieving a 1 in the PUF response. This led to the idea of CHOICE and the concept of the delay-tuning of timing paths to adapt the glitch width $\delta$ by utilizing *Addressable Shift Registers (ASRs)* instead of fixed-size shift registers. This concept is illustrated in Figure 3.4.

Internally, an ASR can be considered a circuit comprising a particular number $S$ of 1-bit memory cells (e.g., programmable flip-flops that can be initialized to either 1 or 0) and a multiplexer (MUX). Figure 3.4 shows an $S$-stage ASR composed of a number $S$ of flip-flops. Here, the input $d$ is used to initialize the internal flip-flops $q_0, \ldots, q_{S-1}$ with a specific bit pattern, denoted as $P$ in the following (not shown in Figure 3.4). Accordingly, the initialization of such an $S$-stage ASR is possible in $S$ clock cycles. After the initialization of the ASR, the address vector $a$ can be used to select which flip-flop determines the output of the ASR. Thereby, each flip-flop itself can be considered as a silicon circuit consisting of two multiplexers and an inverter between its multiplexers, as depicted in the dashed box in Figure 3.4. Although this circuit is synchronous to the clock signal $\text{clk}$, the time required to forward a signal from the input $d$ to the output $Q$ is likewise affected by inevitable propagation delays. For instance, the inverter gate itself has a particular switching time. Thus, the multiplexers will never switch precisely simultaneously, so even between the synchronous flip-flops within the ASR, unavoidable invariances in timing exist.

Typically, an ASR is used in an FPGA to implement variable length shift registers. Since they are available as a hard-macro on the FPGA, this work intends to use them to...
exploit slight differences within the timing paths of the flip-flops between the clock signal (clk) and the ASR's output Q as an unpredictable, freely addressable timing delay. This becomes possible, as each of the flip-flops is fabricated in CMOS technology, which in turn internally comprise MOSFETs, each with slightly different switching characteristics, due to uncontrollable process variations, as introduced and discussed in Section 3.1.2. In other words, if the output of flip-flop q₀ is chosen as the output Q (red path), some signal propagation time $\tau_0$ arise, which is affected at least to some extend by process variations. If one now chooses the output of $q_{S-1}$ instead (cf. green path in Figure 3.4), a different timing path is used, which most likely results in a different signal propagation time $\tau_{S-1}$. Although Xilinx publishes no detailed description of the internal structure of the shift registers, it can be assumed that an ASR of length $S$ provides $S$, not necessarily monotonously increasing but different delays $\tau \in \{\tau_0, \tau_1, \tau_2, \ldots, \tau_{S-1}\}$ depending on the address vector $a$. Using this principle, the tunable PUF design CHOICE instantiates and configures four of these ASRs per PUF bit as presented next.

### 3.2.2 Circuit Design

This subsection describes the implementation and timing details of the PUF circuit, which allows the fine-granular adjustment of the glitch width due to its wide range of configuration options. To this end, Figure 3.5 shows the proposed CHOICE PUF design structure, creating a glitch that arises from delay differences to set the asynchronous preset input (PRE) of the flip-flop. Due to the inherent process variations within the transistors of the PUF circuit and its hard-wired timing paths, it can be assumed that the value of the PUF bit is set randomly. This means that the uncertainty, respectively unpredictability of whether the glitch sets the flip-flop to 1 or not (0), is consistent with the security properties described in Section 3.1.1. The structure and concept of the lightweight CHOICE circuit and its ability to adjust the width of the generated glitch are explained below.

CHOICE utilizes four ASRs, depicted in Figure 3.5 as ASR₀, ..., ASR₃. Here, each ASR controls one corresponding carry chain multiplexer MUX₀, ..., MUX₃. Similar to the Anderson PUF, the logic level on the output $o_{MUX_0}$ of multiplexer MUX₀ determines the value of the asynchronous set input PRE of the shown flip-flop, ultimately representing the PUF bit. However, as seen in Section 3.2.1, a fine-grained tuning of the PUF response becomes possible by exploiting the configurable delay within the ASRs. Each ASR can be individually adjusted via the corresponding address inputs $a_0, \ldots, a_3$ influencing their propagation delay $\tau_{ASR_0}, \ldots, \tau_{ASR_3}$ accordingly. Here, $\tau_{ASR_k}$ denotes the propagation delay (cf. red arrows in Figure 3.5) between a change of the output of the ASR-internal flip-flop $q_0, \ldots, q_{S-1}$ that is selected by the address vector $a_k$ of ASR $k$, $0 \leq k \leq 3$, and its occurrence at the select input of the multiplexer.

Using ASRs of length $S = 32$, the values of $a_0, \ldots, a_3$ can range from 0 to 31. As shown in Section 2.1.1 and Figure 2.3, a CLB within a Xilinx FPGA slice of type SLICEM already provides a total of four ASRs. Nevertheless, the second CLB of the slice is additionally
needed to capture the glitch with the flip-flop. Still, the entire PUF can be implemented in a single slice to generate the glitch, since each slice contains two CLBs.

In Figure 3.5, a choice of two so-called active ASRs is shown in bold. These will be denoted in the following as ASRᵢ, ASRⱼ with \( i, j \in \{0, \ldots, 3\} \) and \( i \neq j \). The two remaining so-called inactive ASRs are initialized such that their outputs do not toggle their multiplexers on each clock cycle and therefore, do not influence the PUF bit generation. Furthermore, let ASRᵢ be the active ASR closest to the flip-flop (uppermost ASR in Figure 3.5), hence \( i < j \). This results in a configuration space of \( \binom{4}{2} = 6 \) pairs of ASRs that are possible to be selected as active, namely: (ASR₀, ASR₁), (ASR₀, ASR₂), (ASR₀, ASR₃), (ASR₁, ASR₂), (ASR₁, ASR₃), and (ASR₂, ASR₃). Together with the 32 delay configurations of each active ASR, the proposed 1-bit CHOICE PUF supports a total of \( \binom{4}{2} \cdot 32^2 = 6,144 \) configurations, each having different timing properties and, consequently, different probabilities of the signature bit to be set to 1. Each PUF configuration can thus be
3.2 CHOICE: A Signature-Tunable PUF for FPGAs

defined by a tuple \( \mathcal{C} = (i, j, a_i, a_j) \) describing the two active ASRs \( i \) and \( j \) and the value of their address inputs \( a_i \) and \( a_j \).

For causing a glitch, each ASR has to be initialized by a specific bit pattern. Equation (3.7) defines the \( S \)-bit initialization pattern \( P(k) \) for each ASR \( k \) with \( k \in \{0, \ldots, 3\} \) depending on the index of the active ASRs \( i \) and \( j \) and their selected addresses \( a_i \) and \( a_j \). Here, inactive ASRs \( (k \notin \{i, j\}) \) are entirely initialized with a 1 sequence, while ASR \( i \) gets initialized with the pattern \( 1\ldots1010 \) in case of a configuration using an even address \( a_j \) and pattern \( 0\ldots0101 \) otherwise. Contrary to ASR \( i \), ASR \( j \) gets initialized with \( 1\ldots1010 \) if \( a_j \) is odd and \( 0\ldots0101 \) when even.

\[
P(k) = \begin{cases} 
1\ldots1111 & \text{if } k \notin \{i, j\}, \\
1\ldots1010 & \text{else} \text{ if } k = i \land a_k \mod 2 = 0, \\
1\ldots1010 & \text{else} \text{ if } k = j \land a_k \mod 2 \neq 0, \\
0\ldots0101 & \text{else.}
\end{cases}
\]

From Eq. (3.7), it can be seen that the 32-bit pattern of the ASRs depends not only on whether the upper multiplexer \( i \) or the lower multiplexer \( j \) is selected but also on the address \( a_k \) of the ASRs themselves. Therefore, if the upper multiplexer with index \( i \) is chosen and the upper ASR has an even address \( a_k \) (indicated in Eq. (3.7) by \( a_k \mod 2 = 0 \)), the pattern \( 1\ldots1010 \) is applied to the upper ASR \( i \) and the pattern \( 0\ldots0101 \) to the lower ASR \( j \). Now, if in a different CHOICE configuration tuple \( \mathcal{C} \), the lower multiplexer is chosen with an odd address \( a_k \) (\( a_k \mod 2 \neq 0 \) in Eq. (3.7)), as well the pattern \( 1\ldots1010 \) needs to be applied, while the upper ASR needs the inverse pattern \( 0\ldots0101 \) for correct toggling. This method requires just 32 clock cycles to load the four ASRs with the corresponding bit pattern \( P \) and consequently explore different CHOICE configurations without changing the circuit itself.

Finally, a \( Z \)-bit signature of the CHOICE PUF design is obtained by instantiating the design illustrated in Figure 3.5 \( Z \) times within the FPGA’s programmable logic. In this context, it should be noted that each such one-bit design is assigned the same configuration tuple \( \mathcal{C} \), both in terms of the choice of the two active ASRs and the address vector values for each ASR. In short, each PUF bit is based on the same circuit configuration, thus no single bit can be manipulated towards a preferred value, while keeping the required hardware resources to a minimum.

3.2.3 Delay Model

The following subsection describes the timing conditions for signal delays that contribute to the modulation of the glitch width. Thus, it eases the understanding of how the tuning affects the PUF response. As mentioned before, the differences, particularly in signal delays, are attributable to different transistor switching characteristics and wire lengths. Even though the timing behavior of transistor switching and the resulting signal
propagation time is affected by process variations, temperature, and supply voltage as described earlier, requirements on switching behavior for glitch generation can accurately be modeled. In this regard, a delay model is presented, which defines the configuration space to adapt a device-specific PUF response as a function of the configuration tuple \( \mathcal{C} = (i, j, a_i, a_j) \) and distinguishes three different switching conditions.

The following signals and delays are considered: First, the two signals \( s_{\text{ASR}_i} \) and \( s_{\text{ASR}_j} \), whose address-tunable propagation delays are determined by the internal delay of the active ASRs, denoted as \( \tau_{\text{ASR}_i} \) and \( \tau_{\text{ASR}_j} \) (cf. red arrows in Figure 3.5). As mentioned in Section 3.2.2, this is referred to as the delay to propagate a signal from the ASR-internal flip-flop output to the select input of the multiplexer once it is triggered by a clock \( \text{clk} \) and an enable signal \( \text{en} \), representing the PUF measurement. The internal propagation delay of the corresponding multiplexers is modeled by times \( \tau_{\text{MUX}_i} \) and \( \tau_{\text{MUX}_j} \) (green labels in Figure 3.5). Furthermore, let \( \tau_i = \tau_{\text{ASR}_i} + \tau_{\text{MUX}_i} \) and \( \tau_j = \tau_{\text{ASR}_j} + \tau_{\text{MUX}_j} \). The third delay describes the time required to forward a signal from the output \( o_{\text{MUX}_k} \) of a multiplexer \( k \), \( 3 \geq k \geq 1 \), to the 1-data input of \( \text{MUX}_{k-1} \) on the carry chain. This delay will be denoted as \( \tau_{\text{CARRY}_k} \) (cf. blue arrows in Figure 3.5) in the following. Depending on the configuration of active ASRs, the sum of delays needs to be considered on the carry chain. Therefore, the propagation delay between the output of multiplexer \( \text{MUX}_j \) and the output of \( \text{MUX}_i \) is denoted as \( \tau_{i,j} \). Since the 1-data input of \( \text{MUX}_3 \) is constantly set to 1 (cf. Figure 3.5), \( \tau_{i,j} \) is the time that the signal \( o_{\text{MUX}_j} \), carrying a 1, is present at the 1-data input of \( \text{MUX}_3 \) before its level changes to 0. In fact, this time depends on the choice of \( \text{MUX}_i \) and \( \text{MUX}_j \) as well as the resulting number of multiplexers in between, since routing length and transistor switching delays will sum up with the distance of multiplexers and thus considerably determine the propagation time. Hence, \( \tau_{i,j} \) can be modeled as follows:

\[
\tau_{i,j} = \sum_{k=i}^{j-1} \tau_{\text{CARRY}_{k+1}} + \tau_{\text{MUX}_k}.
\]  

Based on this model, the question of whether the PUF response is 0 or 1 can be answered by identifying three different delay scenarios. These three scenarios are shown in Figure 3.6, where Figure 3.6a illustrates the first case where the switching of the output signal \( s_{\text{ASR}_i} \) of shift register ASR, occurs faster than the switching of the output signal \( s_{\text{ASR}_j} \) of ASR. In this case, after the propagation delay \( \tau_{\text{MUX}_i} \), \( \text{MUX}_i \) propagates the 1 signal from \( o_{\text{MUX}_i} \) to its output \( o_{\text{MUX}_j} \) until \( \text{MUX}_j \) switches to 0, ending the shown glitch of width \( \delta \). Now, if this glitch is wide enough, the asynchronous input of the flip-flop (cf. PRE Figure 3.5) is set to 1. Since the signal propagation time induced by both the ASR and the multiplexer is crucial for glitch generation, the first scenario can be simplified to the case when \( \tau_i < \tau_j \). Here, the delay \( \tau_{i,j} \) induced by the propagation delay between \( \text{MUX}_j \) and \( \text{MUX}_i \) determines the width \( \delta \) of the glitch.

For the case when \( \tau_i \) is larger than \( \tau_j \), the time \( \tau_{i,j} \) is decisive for whether a glitch occurs at all. This is shown in the second scenario, where Figure 3.6b illustrates the
In the third scenario, $\tau_i$ is still assumed larger than $\tau_j$, but in this case, smaller than the sum of $\tau_{i,j}$ and $\tau_j$. As the 1 signal of $o_{MUX_j}$ is present for the time $\tau_{i,j}$ before it is truncated to 0, an overlap with the switching of MUX$_i$ occurs, which again allows a glitch to appear, only this time most probably not as wide as in the scenario of Figure 3.6a.

In this context, it was observed that such a long propagation of $\tau_{i,j}$ occurs when the active ASR combination (ASR$_0$, ASR$_3$) is chosen since the signal propagation time between the active multiplexers reaches its maximum in this configuration.

According to Figure 3.6, the created glitch width $\delta$ depends on the absolute differences between $\tau_j + \tau_{i,j}$ and the delay $\tau_i$. It follows that the PUF output can be determined according to Eq. (3.9) by comparing $(\tau_j + \tau_{i,j}) - \tau_i$ with a minimum glitch width $\delta_{\text{min}}$:

$$PUF = \begin{cases} 1 & (\tau_j + \tau_{i,j}) - \tau_i \geq \delta_{\text{min}}, \\ 0 & (\tau_j + \tau_{i,j}) - \tau_i < \delta_{\text{min}}. \end{cases} \quad (3.9)$$

Such a minimum glitch width is required to set the PUF bit to 1 since the routing path from the signal $o_{MUX_0}$ to the flip-flop behaves like a low-pass filter due to the ohmic resistance $R_w$ and capacitive load $C_w$ of the wire $w$. The resulting time constant
\( \tau \propto R_{w}C_{w} \) depends naturally on the length of the wire, in this case, the absolute distance between the output of the upper multiplexer MUX\(_0\) and the preset input of the flip-flop. As a result, high-frequency glitches are filtered out, keeping the PUF bit at 0 even when a glitch occurs. However, on the other hand, a created glitch \( \delta \) that is wider than \( \delta_{\text{min}} \) increases the probability that the glitch reaches the flip-flop's preset input and sets the PUF bit to 1. It can be deduced that adjusting the glitch width \( \delta \) by the address-tunable propagation delays, as introduced in this section, will increase or decrease the probability of obtaining a 1 in the PUF response.

In the following Section 3.3, it is shown how the configuration of both ASRs and their pairing with the carry chain determines the outcome of the PUF circuit. For this reason, the following section provides the experimental evaluation of the proposed CHOICE PUF architecture and demonstrates how its configuration space can be tuned to optimize the statistical properties of uniqueness, reliability, and robustness and to counteract the mentioned effects of changing environmental conditions.

### 3.3 Experimental Evaluation

This section presents the results of a series of experiments conducted to evaluate the impact of CHOICE's tunability on the security characteristics of the obtained PUF responses. For assessing the PUF architecture, a Xilinx Zynq xc7z010clg400-1 Programmable System-on-Chip (PSoC) featuring a 28 nm FPGA and ARM-based processor system, as presented in Section 2.1.1, was used. \( N = 6 \) of these PSoC chips, found on the Digilent Zybo evaluation board were examined to evaluate the PUF responses and security properties. A specific instance of these boards is denoted by \( b \) with \( b \in \{B0, \ldots, B5\} \). In addition to the fundamental PUF security properties of uniqueness and reliability introduced in Section 3.1.1, two additional indicators are established to assess the PUF's tunability and resilience to high-temperature variations for this evaluation.

For this purpose, a hardware/software co-design was developed on the PSoC mentioned above. This co-design allows the configuration of the PUF and readout routines to be performed in software on the central processing system of the PSoC. The floorplan of this co-design is illustrated in Figure 3.7. It details the placement of a 128-bit (\( Z = 128 \)) CHOICE PUF depicted as orange cells in the upper left region (X0Y1) of the FPGA. The response length of 128-bit for evaluation was chosen because this corresponds to the key size of an AES module, as discussed in Section 2.3.1, but is also an appropriate length for secure device authentication. In this regard, a dedicated area, referred to as PUF_Block_Zone in Figure 3.7, isolates the PUF from additional required logic (cyan cells in Figure 3.7), which is arranged with appropriate spacing around the PUF zone. The additional logic is required to automate the configuration of the ASRs, triggering of PUF measurements, and the readout of the resulting responses via memory-mapped HW/SW interface. This automation allows exploring all 6,144 different tuning options, performing fast PUF configuration and measurement routines. Meanwhile, a host-side...
3.3 Experimental Evaluation

Figure 3.7: Floorplan of a Xilinx Zynq xc7z010clg400-1 PSoC device showing the placement of a \( Z = 128 \)-bit CHOICE PUF (orange cells) in a separate area of the FPGA (PUF_Block_Zone) for PUF response evaluation. The PUF is surrounded by additional logic (cyan cells) to allow automated configuration and readout routines performed in software on the processor system of the PSoC (orange block on the left), forming a hardware/software co-design.

application can be connected to this application to capture the measured PUF data from the connected devices. In this way, multiple devices can be evaluated simultaneously and evaluated throughout all configurations.

This measurement setup is only required for the PUF configuration and tunability evaluation described in the following subsections. After exploring and evaluating all configuration options, a single final configuration, along with the required PUF circuit can be provided e.g., as an IP block or a netlist. In this case, an external configuration interface to the host side is no longer required. It is also true that even if the chosen configuration of the CHOICE PUF should be compromised, the circuit cannot be cloned because it has been adjusted to achieve a certain level of uniqueness, as will be explained below. To this end, the tunability effectiveness is evaluated and compared to the already existing silicon PUFs based on non-adaptable delay circuits introduced in Section 3.1.4.
The remaining section is organized as follows. Section 3.3.1 demonstrates the tunability of the PUF by introducing the indicator of uniqueness to quantify in which range the PUF can be tuned. Subsequently, the results of the uniqueness and reliability investigation are presented in Section 3.3.2 and Section 3.3.3, respectively. For this purpose, both formulas in Eq. (3.11) and Eq. (3.12) are extended to account not only for the influence of a particular board \( b \) but also for that of a particular CHOICE configuration option \( c \). Thereafter, in Section 3.3.4, the impact of temperature on the PUF response is discussed by establishing a novel indicator to evaluate the robustness of a CHOICE configuration. In this investigation, it will be shown that tuning is beneficial in finding particularly temperature-robust configurations. At this point, it is important to note that for each board \( b \in \{B0, \ldots, B5\} \) and configuration option \( c \in [0 : 6, 144 - 1] \), \( M = 1,000 \) independent measurements were performed with each measurement \( m \in [0 : M - 1] \) delivering a response \( r_{b,c}^m \in \{0, 1\}^Z \), with \( Z = 128 \)-bit. This way, it is possible to ensure that the acquired data has correspondingly strong validity.

### 3.3.1 Uniformity

A *uniformity* indicator is calculated for each configuration on each board to see in what range a CHOICE PUF response can be tuned. The uniformity evaluates the proportion of zeros and ones in the nominal PUF response \( r_{b,c}^* \in \{0, 1\}^Z \) for each configuration \( c \) and board \( b \) over the response length \( Z \), according to the following formula:

\[
\text{Uniformity}_{b,c} = \frac{1}{Z} \cdot \sum_{i=0}^{Z-1} r_{b,c}^* (i) \cdot 100\% ,
\]

where \( r_{b,c}^* (i) \) corresponds to the \( i \)-th bit of the respective nominal response while uniformity does not allow a direct conclusion on the security of the PUF, it is still an indicator of the randomness of the response. More precisely, 0% uniformity means that no bit is set to one in the response, while 100% means that every PUF bit has a value of one. Consequently, a value of around 50% implies an equal number of ones and zeros in the nominal response, and thus, is a good indicator for high randomness of the PUF.

For this reason, the first evaluation examines how tuning affects the uniformity of the PUF. In this regard, Figure 3.8 presents the uniformity value for each of the 6,144 different configurations. Here, each configuration for each board \( b \in \{B0, \ldots, B5\} \) was sorted, based on its uniformity value in percent. From the curves it can be seen that it is possible to tune a PUF response within the entire uniformity range of 0 to 100% across all six boards under test. Even if the exact sequence of individual configurations could slightly differ, the curves demonstrate that the tuning behaves consistently among all boards, highlighting the general applicability of the tuning. Moreover, this validates the modeling of delay adaptation introduced in Eq. (3.9) since the probability of obtaining a one or a zero in the PUF response can be controlled across different configuration levels.
3.3 Experimental Evaluation

Figure 3.8: CHOICE PUF response, evaluated for uniformity on six different FPGA boards (B0–B5) across all 6,144 different configuration levels. As can be seen, configurations can be found for all boards to produce PUF responses in a uniformity range between 0% to 100%. Highlighted in green is also the interval of configurations leading to the highest uniqueness (see also Figure 3.9) [SKB²¹a].

This provides some essential key advantages over static PUF designs. As will be shown in the following, this positively affects the average uniqueness of PUF responses, especially in a uniformity range of 30-70% (cf. green interval in Figure 3.9). Another advantage is that, as it will be shown in Section 3.3.3, that within this green sector, at least one configuration exists with a BER below 1.5% for each device. Last but not least, from this range of configurations with sufficient uniqueness, configurations that are particularly temperature-stable can be found.

3.3.2 Uniqueness

Due to the tunability of CHOICE, the indicator of uniqueness as introduced in Section 3.1.1 needs to be extended to account for the different configuration levels $c$ as well. Therefore, uniqueness is reformulated, in this work, as the average *inter-die uniqueness per configuration level*, which can be calculated across all boards for each
configuration according to the following formula:

\[
\text{Uniqueness}_c = \frac{2}{N \cdot (N - 1)} \cdot \frac{1}{Z} \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} HD(r^*_i, r^*_j) \cdot 100\%.
\] (3.11)

With \( i, j \in \{B0, \ldots, B5\} \), the result of the uniqueness assessment across all six boards investigated for each of the \( c = 6,144 \) configurations is shown in Figure 3.9. As can be seen from the combination of Figure 3.8 and Figure 3.9, a strong uniqueness of about 50% follows from tuning the uniformity towards a range between 30% and 70% (cf. green interval in Figure 3.8), as this interval can be mirrored to the green sector in Figure 3.9. In fact, within this green sector, 147 different configurations were found between a close to optimal 49% to 51% uniqueness. In the following subsection, the reliability of these configurations with extraordinary uniqueness is presented and compared to related silicon PUFs, as introduced in Section 3.1.4.
3.3.3 Reliability

Similar to the uniqueness, the reliability indicator for silicon PUFs, initially introduced in Section 3.1.1, needs to be reformulated for the tunability of CHOICE. Specifically, the reliability of each configuration can be restated to the Bit Error Rate (BER) of a configuration $c$ on board $b$ over all $M$ measurements by the Hamming distance $HD$ from the nominal value $r^*_b \in \{0, 1\}^Z$ as follows:

$$\text{BER}_{b,c} = \frac{1}{M} \cdot \sum_{m=0}^{M-1} \frac{HD(r^*_b, r^m_{b,c})}{Z} \cdot 100\%.$$  \hspace{1cm} (3.12)

In this context, the trade-off between reliability and randomness was mentioned in Section 3.1.1, where responses with a low BER tend to have more insufficient randomness, and thus often poor uniqueness. CHOICE provides a solution in this regard by providing not just one configuration, as with static PUFs, but multiple configurations that already satisfy the uniqueness requirements. These configurations can then be further examined with respect to their reliability in order to subsequently select only those that additionally also achieve a low BER.

The advantage of this procedure becomes evident when considering Figure 3.10. The diagram shows all 147 configurations from the previous evaluations (Figure 3.8 and Figure 3.9), providing an average uniqueness between 49 and 51% (blue line). Meanwhile, the average BER (red line) varies between a low 2.2% and 3.8% for all these configurations. A closer look shows that 13 configurations even provide an average BER of less than or equal 2.5%. In this context, configuration 32 ($\mathcal{C} = (0, 3, 26, 7)$) has the lowest average BER of only 2.2% across all boards while simultaneously providing a uniqueness of 49.22%.

A direct comparison with the individual security characteristics of the silicon PUFs presented in Section 3.1.4 makes this advantage even better visible. The tunability makes it for the first time possible to provide a device-specific PUF setup by choosing the configurations that produce the lowest BER for a given board, as shown by the colored triangles in Figure 3.10. Here, the dashed lines indicate the board- and configuration-specific bit error rate, $\text{BER}_{b,c}$ over the 147 configurations with a uniqueness between 49% and 51%, while the colored triangles mark their found minimum.

Another advantage of such a device-specific PUF setup is that individual device conditions can be addressed. Consequently, this would account for the aging and temperature effects described in Section 3.1.3 to achieve better BER characteristics than a single fixed PUF implementation. The results shown in Table 3.1 demonstrate this. Table 3.1 shows the comparison of the security-related PUF properties and resource requirements of CHOICE with the PUF architectures presented in Section 3.1.4. It can be seen that the chosen configurations can compete with the PUF in [ZWD+16] in terms of uniqueness. Moreover, CHOICE is close to the achieved average BER of [UKM+18] but at the same time outperforms all approaches in terms of slices required per PUF bit and BER
when compared to the board-specific BERs $\text{BER}_{b,c}$. Here, even a remarkable minimum of only 0.97% BER was found on board $B3$. Concerning the non-glitch-based PUFs, only the 3-XOR Arbiter PUF from [MYI′15] can compete with a BER of 1.18%. However, its low uniqueness value could potentially make the PUF predictable. Nevertheless, all these reliability measurements were carried out under stable temperature conditions. For this reason, in the following subsection, the robustness of CHOICE under the influence of heavy temperature differences is investigated and compared against the presented related work approaches.

### 3.3.4 Temperature Robustness

The following subsection comprehensively analyzes how the temperature effects introduced in Section 3.1.3 may affect PUF responses and shows how the tunability of CHOICE is beneficial to determine configurations with a high temperature robustness.

---

$^6$only a 64-bit ($Z = 64$) PUF
According to the previous evaluations in Section 3.3.1, Section 3.3.2, and Section 3.3.3, temperature robust configurations were found that simultaneously satisfy the requirements for uniformity, uniqueness, and reliability.

In this context, an ambient temperature range between 0 °C and 70 °C was chosen to achieve a device operating temperature range of \( \Delta T = 70 \) K. Thereby, the temperature dependence \( T_j \approx T_A + 15 \) K could be measured across all boards for the investigated temperature interval. This dependency lead correspondingly to a measured junction temperature \( T_j \) (in-chip) between 15 °C to 85 °C, marking the maximum chip operating temperature specified by the manufacturer for commercial-grade devices [Xil20].

In the following, the indicator of temperature robustness is defined and evaluated for five different ambient temperatures in a temperature vector \( T_A = \{0 \, ^\circ\text{C}, 20 \, ^\circ\text{C}, 40 \, ^\circ\text{C}, 60 \, ^\circ\text{C}, 70 \, ^\circ\text{C}\} \), where 20 °C is the reference temperature to compare. Again, \( M = 1,000 \) independent measurements are taken for each configuration option \( c \in \{0 : 6, 144 - 1\} \) on each board \( b \in \{B0, \ldots, B5\} \). This time, however, the measurements \( m \in \{0 : M - 1\} \) are also repeated for each of the specified ambient temperature values \( T_A^{(i)} \in T_A \). In this context, first, the uniformity (cf. Eq. (3.10)) of the PUF response is inspected to see how the temperature affects the glitch generation on each board. Figure 3.11 shows the uniformity of the 6,144 different configurations on two selected boards, namely B3 and

<table>
<thead>
<tr>
<th>PUF Properties</th>
<th>PUF</th>
<th>Slices</th>
<th>Bit</th>
<th>Uniqueness ( c )</th>
<th>avg. BER</th>
<th>BER ( b,c )</th>
<th>( i,j )</th>
<th>( a_i )</th>
<th>( a_j )</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUF</td>
<td>Slices</td>
<td>Bit</td>
<td>Uniqueness ( c )</td>
<td>avg. BER</td>
<td>BER ( b,c )</td>
<td>( i,j )</td>
<td>( a_i )</td>
<td>( a_j )</td>
<td></td>
</tr>
<tr>
<td>CHOICE on boards ( b \in {B0, \ldots, B5} )</td>
<td>1</td>
<td>49.17 %</td>
<td>2.71 %</td>
<td>1.27 %</td>
<td>1.3</td>
<td>25</td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>50.57 %</td>
<td>2.39 %</td>
<td>1.39 %</td>
<td>0.2</td>
<td>25</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>49.06 %</td>
<td>2.96 %</td>
<td>1.06 %</td>
<td>1.2</td>
<td>11</td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>50.88 %</td>
<td>2.75 %</td>
<td>0.97 %</td>
<td>1.2</td>
<td>0</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>50.47 %</td>
<td>2.81 %</td>
<td>1.15 %</td>
<td>1.2</td>
<td>18</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>50.21 %</td>
<td>2.37 %</td>
<td>1.44 %</td>
<td>1.2</td>
<td>28</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usmani et al. [UKM+18]</td>
<td>2</td>
<td>46.25 %</td>
<td>2.39 %</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>Zhang et al. [ZWD+16]</td>
<td>4</td>
<td>49.68 %</td>
<td>3.17 %</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>Anderson [Andi0]</td>
<td>2</td>
<td>48.28 %</td>
<td>N/A</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>3-XOR Arbiter [MYI+15]</td>
<td>3</td>
<td>54.96 %</td>
<td>1.18 %</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>RO [MSI1]</td>
<td>4</td>
<td>45.51 %</td>
<td>3.21 %</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>Butterfly [KGM+08]</td>
<td>2</td>
<td>43.16 %</td>
<td>3.80 %</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: CHOICE resource requirements (Slices per Bit) as well as uniqueness and BER over six investigated PSoC boards with corresponding configuration settings [SKB*21a] in comparison to already existing glitch-based PUFs [UKM*18; ZWD*16; Andi0] and other related-work [MYI*15; MSI1; KGM*08].
3 Cryptographic Key Provisioning using Physical Unclonable Functions

Figure 3.11: CHOICE PUF response evaluated on two boards for uniformity under five different ambient temperature conditions (0 °C–70 °C) across all 6,144 different configuration levels. As can be seen, temperature affects the PUF response on boards differently, while board B3 shows only a low variation in uniformity on different temperatures, board B1 is more sensitive to temperature variation. Despite the temperature effects, configurations can be found to produce PUF responses in a uniformity range between 0% to 100% [SKB*21b].

B1, where the configurations were again first sorted by their uniformity value for each temperature.

As can be seen from the curves, a high temperature, e.g., 70 °C, affects the PUF response so that fewer ones are generated. Consequently, a low temperature, e.g., 0 °C, increases the probability of observing ones in the response. Moreover, as can be seen in the comparison of Figure 3.11a and Figure 3.11b, uniformity is also an initial indicator of how sensitive the PUF is to temperature variations when placed on different silicon devices. For instance, board B3 shows only a slight deviation in uniformity between the highest and the lowest temperature. In contrast, board B1 is more sensitive to temperature variations, as can be seen from the strongly diverging uniformity curves. It can be assumed that different operating times, a certain degree of stress, and other factors that cause accelerated aging are responsible for these effects visible in the measurements.

However, in order to quantify the reliability of individual configurations at different ambient temperatures, the BER of configuration c on board b is calculated as the average Hamming distance HD over all M measurements from the nominal value $r_{b,c,T_A}^*$ at
20 °C. Thereby, only configurations from Section 3.3.3 were considered, which already revealed an excellent uniqueness between 49% and 51% at a nominal temperature of 20 °C. In order to rank the configurations according to their robustness, the indicator of temperature robustness is defined as follows. A configuration \( c \) of a board \( b \) is considered robust if the variation of the bit error rates \( \text{BER}_{b,c,T_A}^{(i)} \) for the different temperatures \( T_A^{(i)} \) shows a minimum difference. Robustness is therefore calculated as the piece-wise absolute difference of the bit error rates between the consecutive temperature settings \( T_A^{(i)} \) and \( T_A^{(i+1)} \) divided by their temperature difference through the following formula:

\[
\text{TR Robustness}_{b,c} = \left| \frac{\sum_{i=0}^{|T_A|-2} |\text{BER}_{b,c,T_A}^{(i)} - \text{BER}_{b,c,T_A}^{(i+1)}|}{T_A^{(i)} - T_A^{(i+1)}} \right|, \tag{3.13}
\]

where \( \text{BER}_{b,c,T_A}^{(i)} \) corresponds to the average bit error rate occurring at the \( i \)-th ambient temperature within the vector \( T_A \) compared to the nominal response selected for the same configuration and board at 20 °C. In this way, a PUF is considered resilient to environmental stress if it exhibits a low bit error rate over significant temperature differences. Hence, the PUF performs robustly against harsh operating conditions, leading to a robustness value close to 0 in the end.

The result of the robustness evaluation revealed that 108 out of 147 configurations (cf. Figure 3.10) never exceed a BER of 8% over the entire ambient temperature range of 0 °C to 70 °C on all boards under test. Furthermore, CHOICE allows finding at least one configuration on each board where, on average, no more than 8% of the bits flip. Following Eq. (3.13), these particular robust configurations are presented in Figure 3.12. It is evident from Figure 3.12 that even the configurations with the best temperature stability are still sensitive to temperature variations. Obviously, the more the temperature approaches the limits of the specified operating temperature, the more difficult it becomes to reproduce the nominal PUF response \( r^*_{b,c} \in \{0,1\}^Z \). Nevertheless, the best temperature-stable configuration found does not even exceed the 3% mark with a BER of only 2.71% at 70 °C on board B3.

The observation shown in Figure 3.11 reflects that the susceptibility to temperature variations depends mainly on the actual constitution of the silicon device on which the PUF is implemented. Nonetheless, this is just further evidence of the need for tunable PUFs, since only a board-specific configuration can achieve sufficient robustness to temperature variations. The corresponding configurations of such robust board-specific setup are listed in Table 3.2. Here, the best configuration for each board is shown in terms of its uniqueness, robustness, and average and maximum BERs over the temperature range studied, along with a comparison to the glitch-based work presented in Section 3.1.4. In this context, Table 3.2 further highlights that CHOICE’s tunability is beneficial to maintain its already-strong reliability properties despite a temperature range of 70 K. Moreover, this is true even when compared to existing approaches, although some have only investigated temperature ranges of 40 K or even less.

Considering the unique, reliable, and temperature-robust nature of the CHOICE
Cryptographic Key Provisioning using Physical Unclonable Functions

![Graph](image)

**Figure 3.12:** Best found device-specific PUF configurations in terms of BER, which illustrates the temperature robustness over a wide temperature range of 70 K. At least one configuration exists for each board that does not exceed a BER of 8%. [SKB*21b].

PUF, the proposed approach is ideally suited for cryptographic key generation. Such a PUF-based cryptographic key generation is presented below, along with its storage and performance overheads.

**Table 3.2:** Most temperature robust CHOICE configurations found on six investigated PSoC boards for a $\Delta T = 70 K$ ambient temperature change with uniqueness and BER characteristics compared to the glitch-based PUFs of Section 3.1.4 [SKB*21b].

<table>
<thead>
<tr>
<th>PUF Properties</th>
<th>Config. $\mathcal{C}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUF</td>
<td>Uniqu., Robust, avg. BER, $\max_{T_A} { BER_{b,c,T_A} }$, ${ i, j, a_i, a_j }$</td>
</tr>
<tr>
<td>CHOICE on boards $b \in { B_0, \ldots, B_5 }$</td>
<td>$\Delta T = 70 K$</td>
</tr>
<tr>
<td></td>
<td>49.53 % 0.31 4.22 % 5.79 % 1, 3 17 15</td>
</tr>
<tr>
<td></td>
<td>50.05 % 0.60 4.36 % 7.69 % 1, 3 2 3</td>
</tr>
<tr>
<td></td>
<td>49.21 % 0.30 3.23 % 4.74 % 0, 3 16 31</td>
</tr>
<tr>
<td></td>
<td>49.48 % 0.11 2.15 % 2.72 % 1, 2 27 27</td>
</tr>
<tr>
<td></td>
<td>49.16 % 0.43 3.42 % 6.17 % 1, 3 23 7</td>
</tr>
<tr>
<td></td>
<td>50.21 % 0.38 3.31 % 4.89 % 1, 3 2 7</td>
</tr>
<tr>
<td>Usmani et al. [UKM*18] at $\Delta T = 40 K$</td>
<td>N/A N/A 4.83 % 11.24 % / / /</td>
</tr>
<tr>
<td>Zhang et al. [ZWD*16] at $\Delta T = 30 K$</td>
<td>45.67 % N/A 9.25 % 18.75 % / / /</td>
</tr>
<tr>
<td>Anderson [And10] at $\Delta T = 30 K$</td>
<td>N/A N/A 3.6 % 7.81 % / / /</td>
</tr>
</tbody>
</table>
In this section, a case study is presented and evaluated in which CHOICE is used to generate a secure, device-specific cryptographic key from a noisy PUF response. As introduced in Section 2.3, trusted keys are required for a variety of cryptographic algorithms, causing their security to rely on the confidentiality of these keys. In this context, PUFs provide a promising solution to ensure the secure provisioning of such keys. In this way, the PUF becomes a trust anchor on which highly system-critical processes, such as secure boot (presented in Chapter 4) or remote updates (presented in Chapter 5), can be built.

As explained in Section 2.3.1, a key used, e.g., for AES, must meet specific requirements to be accepted as a cryptographic one. In addition to fundamental security attributes such as key length and uniqueness, the key must be generated randomly, likewise to a PUF response, so that the key is unpredictable. Since the evaluation of uniqueness has already shown that CHOICE provides this randomness across all boards (see Section 3.3.2), only the stability of the PUF response needs to be guaranteed to derive a reliable key. Therefore, this section proposes an approach for secure key generation from a noisy PUF output. More specifically, Section 3.4.1 first examines the impact of majority voting to reduce dynamic noise in the PUF response before introducing fuzzy extraction as a method for error correction in Section 3.4.2. A concrete design and evaluation of the resulting PUF-based key generator follow in Section 3.4.3.

### 3.4.1 Majority Voting

Previous work [XH16; AMS+10] has shown that majority voting is beneficial to reduce dynamic noise in PUF responses. For instance, for an odd-number \( M_v \) of CHOICE PUF measurements, where each measurement \( m_v \in [0 : M_v - 1] \) yields a response \( r_{b,c}^{m_v} \in \{0, 1\}^Z \) (called a candidate), majority voting can be performed for each bit within the response. For a single bit, this involves voting for a one if more than \( m_v = \frac{M_v - 1}{2} \) ones were observed across all measurements \( M_v \). Correspondingly, a zero is determined for that bit if fewer than \( m_v \) ones were measured. While this voting process improves the reliability of bits that become unstable under external influences, such as a changing temperature, this comes at the price of making the PUF evaluation take longer since \( M_v - 1 \) measurements are required. The CHOICE PUF response resulting from such voting for a given configuration \( c \) and board \( b \) will be denoted \( r_{b,c}^i \in \{0, 1\}^Z \) of length \( Z \) in the following.

In the context of this work, majority voting is applied to investigate temperature-induced outliers in the PUF response and to improve its stability. For this purpose, two configurations from Figure 3.12 that exceed a 6% BER value are selected for evaluation, namely \( C = (1, 3, 2, 3) \) on board \( B1 \) and \( C = (1, 3, 23, 7) \) on board \( B4 \). A comparison was then made between the voted response \( r_{b,c}^i \) and the nominal response \( r_{b,c,T}^{(i)} \) at
20 °C based on the Hamming distance $HD$. Next, the number of votes respectively measurements $M_v$ needed to push $\max_{T_A \in T_A} \{BER_{b,c,T_A}\}$ below a limit of 6% BER was investigated. Here, the error correction capability of 6% was chosen, as it represents a good compromise [HMS*12] between implementation effort and correctability in case of subsequent error correction (see also Section 3.4.2).

As a result of this investigation, for configuration $C = (1, 3, 2, 3)$ on board $B1$, it can be stated that the bit flips manifest themselves at extreme temperatures in this configuration since it was not possible to reduce the $\max_{T_A \in T_A} \{BER_{b,c,T_A}\}$ of 7.69% below an error of 6% even with $M_v > 100$ votes. In contrast, however, it was determined that only $M_v = 15$ measurements were required to achieve a constant $\max_{T_A \in T_A} \{BER_{b,c,T_A}\}$ below 6%, i.e., 5.86%, for configuration $C = (1, 3, 23, 7)$ on board $B4$. This observation is depicted in Figure 3.13, showing a direct comparison of error percentages without voting in Figure 3.13a and with voting in Figure 3.13b, respectively. In Figure 3.13a one can see how the high number of outliers negatively affect the PUF response without voting, resulting in an avg. $BER_{b,c,T_A}$ of 3.42% and a $\max_{T_A \in T_A} \{BER_{b,c,T_A}\}$ of 6.17%. In contrast, outliers are almost entirely removed when voting over $M_v = 15$ measurements, leading to a significant improvement in the avg. $BER_{b,c,T_A}$ of 2.66% and $\max_{T_A \in T_A} \{BER_{b,c,T_A}\}$ of 5.86%.

While this subsection has shown that majority voting can significantly improve reliability at each temperature point, additional error correction is required to obtain a unique static secret, as discussed below.

3.4.2 Fuzzy Extraction

As can be seen from the previous sections, PUF-based cryptographic key generation requires information reconciliation [DGS*14], respectively, error correction to derive a singular static secret from a noisy PUF response. In this context, a fuzzy extractor facilitates the conversion of the noisy data from a secret $w$, denoted $w'$ in the following, into a uniformly distributed cryptographic key $k \in \{0, 1\}^Z$ of length $Z$. In this regard, the security foundation of the methods proposed in the following chapters (i.e., Chapter 4 & Chapter 5) is based on the cryptographic key $k \in \{0, 1\}^Z$ generated from a CHOICE PUF by such a fuzzy extractor together with required helper data [DGS*14], denoted as $H$ in the following. This subsection applies the concept of a reusable fuzzy extractor [Boy04; CFP*16] to CHOICE, which enables cryptographic key generation within the programmable hardware of the FPGA.

As introduced in Section 2.3.3, the difference between secret data $w$ and a cryptographic key $k \in \{0, 1\}^Z$ is that the secret $w$ is not directly useable for cryptographic operations. This might be, for instance, because the length of the secret is simply insufficient or because the secret is not a uniform random number. The same is correspondingly valid for the noisy reading $w'$ of a secret $w$. In this case, dedicated helper data are required to derive a cryptographic key $k \in \{0, 1\}^Z$ with corresponding length $Z$ from
such a noisy reading of the secret $\mathbf{w}'$, which cannot be distinguished from a uniform random key.

Generating a cryptographic key from a noisy secret $\mathbf{w}'$ and helper data $\mathcal{H}$ requires two distinct phases [DGS+14]. Public helper data is generated from the secret $\mathbf{w}$ in a secure environment during the so-called enrollment phase in a first step. After generation, it is safe to store the helper data unencrypted on the device’s non-volatile memory, even in an untrusted environment. During the reconstruction phase, the cryptographic key $\mathbf{k} \in \{0,1\}^Z$ can be recovered in a second step from a noisy secret measurement, using the helper data $\mathcal{H}$ available in the device’s non-volatile memory. However, key generation is only successful in this phase if the noisy secret is within a certain error limit, typically specified as the number of tolerable bit errors based on the Hamming distance $H_D$. In
particular, a successful generation requires that the secret \( w \) be reconstructed by the fuzzy extractor, if and only if \( w' \) does not deviate from \( w \) by more than \( HD \) bits.

For the enrollment phase, this work generates a list of public helper data from a nominal CHOICE PUF response \( r^*_{b,c,T_A} \) at 20 °C for a given board \( b \) and configuration \( c \), representing the FPGA-inherent underlying secret \( w \). In this context, the amount of non-volatile memory required to provide sufficient helper data in the field depends on several factors, which are discussed next. First, the helper data consists of a minimum number of entries required to reconstruct the key from a noisy measurement. The number of entries \( E \) depend on the number of tolerable errors \( HD \) in the measurement, which are expressed in [CFP+16] as follows:

\[
E = e^{HD \cdot \log_2 \left( \frac{2}{\gamma} \right)}.
\]  

Here, the parameter \( \gamma \in [0, 1] \) determines the probability that repeated helper data generation from the same nominal response result in the same key \( k \in \{0, 1\}^Z \) during the enrollment phase. This parameter allows the enhanced security property of a reusable fuzzy extractor, initially proposed by Canetti et al. in [CFP+16]. From this, it can be stated that a maximum Hamming error of \( HD \) bits is tolerable while the original key can be regenerated with probability \((1 - \gamma) \cdot 100\%\) if the following holds. Let \( \gamma \) be a considerably small value, e.g., 0.001. Let \( r^*_{b,c} \) further be a noisy PUF response that differs in a maximum number \( HD \) of bits, e.g., \( HD = 2 \), from a nominal PUF response \( r^*_{b,c,T_A} \) at 20 °C. Then, according to Eq. (3.14), a unique cryptographic key is generatable with a probability of 99.9% if at least \( E = 81 \) entries of helper data are provided. Thus, the corresponding memory requirement, denoted \( S_H \), can be determined as a function of \( E \) according to the following formula:

\[
S_H (E) = E \cdot (2 \cdot Z + R + Y),
\]  

where \( Z \) denotes the size of the derived cryptographic key \( k \in \{0, 1\}^Z \), \( R \) the size of random data \( S \) (i.e., a salt, cf. Section 2.3.3), and \( Y \) the size of a required extraction indicator, denoted as \( \alpha \) in the following. Accordingly, for a PUF response of \( Z = 128 \)-bit, the helper data \( \mathcal{H} \) storage requirements can be calculated from Eq. (3.15), where \( Z \) and \( R \) are 16 bytes in size, while \( Y \) requires two additional bytes of zero-initialized storage. Now that the fundamentals of reusable fuzzy extractors have been covered, the following subsection presents their implementation in hardware and discusses performance and cost implications for cryptographic key generation.

### 3.4.3 Design and Evaluation of a PUF-Based Key Generator

The helper data created during the enrollment phase can be generated in software on a processor since this is to be done in a secure environment. In contrast, the reconciliation phase for cryptographic key generation from a noisy PUF measurement is done in a
Figure 3.14: Schematic representation of the implemented PUF-based key generation scheme, leveraging dynamic noise cancellation in the PUF response $r_{bo} \in \{0, 1\}^Z$ through temporal majority voting and subsequent error correction via fuzzy extraction and helper data to derive a cryptographic key $k \in \{0, 1\}^Z$ [SKB’21b].

potentially insecure environment. However, if the reconciliation phase is performed in hardware within the FPGA’s programmable logic, the generated key stays only within the FPGA and can thus, be considered more secure.

In the following, a dedicated hardware design is proposed to quantify the required hardware resource costs for such PUF-based key generation. This hardware design performs temporal majority voting based on a Boyer-Moore voting scheme [BM91] combined with fuzzy extraction described by Canetti et al. in [CFP’16]. Figure 3.14 gives an overview of the necessary building blocks of the PUF-based key generator defined here. The key generator is configured to trigger multiple consecutive CHOICE PUF measurements $r_{bo} \in \{0, 1\}^Z$ on a given board $b$ and configuration $c$. During the measurements already, temporal majority voting is performed to obtain a potentially noise-reduced response $r_{bo}$ as described in Section 3.4.1. The fuzzy extractor then iterates at most $E$ times over the provided list of helper data $\mathcal{H}$. This iteration involves processing the helper data, consisting of a list of random salts $S$, bitmasks $M$, and Keyed-Hash Message Authentication Code (HMAC) ciphers $C$ by a Password-Based Key Derivation Function (PBKDF) module, as introduced in Section 2.3.3. For key recovery, the output of the PBKDF module, denoted in Figure 3.14 as $DK$, is XOR-linked with the cipher elements $C$ at each iteration step, resulting in an extended key $K$. The two-byte extraction indicator $\alpha$, indicates the recovered cryptographic key $k \in \{0, 1\}^Z$ in the extended key $K$ if the two least significant bytes of the extended key are zero before the entire list of helper data has been processed. A detailed algorithmic description of the key extraction process follows in Section 4.2.2, as this method enables on-the-fly key generation during the proposed secure boot process.

The Xilinx Zynq PSoC mentioned above (see Section 3.3) was chosen to implement and test this key generation scheme for a key size of $Z = 128$-bit, $M = 15$ votes, and a PBKDF function based on the SHA-256 hash function. Here, the design objectives performance and resource costs were investigated at a 10 ns clock cycle time. Table 3.3 shows the number of FPGA resources required for the cryptographic key generation in this study in terms of FFs, LUTs, and 36 kbit BRAM – no DSPs slices were needed. With 8% of LUTs and 7% of FFs on the second smallest PSoC from the Zynq family, the numbers
indicate that a PUF-based cryptographic key generation is even feasible on entry-level PSoC devices. Moreover, since the required hardware overhead remained constant regardless of the key size $Z$ and expected Hamming error $HD$, resource requirements are not affected even for larger keys, e.g., $Z = 256$ bits.

As can be seen from Eq. (3.14) and Eq. (3.15), the storage requirements of the helper data depend on the number of errors $HD$ to be corrected. This leads to a trade-off between the error correction capability and implementation complexity, particularly in terms of helper data size [HMS+12]. Therefore, the maximum processing time in clock cycles and the helper data storage requirements $S_H$ in bytes are evaluated next for a $Z = 128$-bit CHOICE PUF cryptographic key. In particular, the best board-specific configurations found in Table 3.2 were tested for their $\max_{T_k \in T_A} \{ \text{BER}_{b_k,T_k} \}$. This testing involved generating helper data for error correction in a range of $HD \in \{1, 2, 4, 6, 8\}$, which corresponds to a $\max_{T_k \in T_A} \{ \text{BER}_{b_k,T_k} \}$ of 0.78%, 1.56%, 3.13%, 4.69%, respectively, 6.25% for a key with $Z = 128$-bit.

In addition to the storage requirements for helper data, the Worst-Case Execution Time (WCET) was used as a performance indicator to reconstruct the key. The WCET is a good fit as traversing the entire list of helper data represents the worst case for successful key generation, so to say, the key is recovered from the last element of the list of helper data. For this, $\gamma$ was first set to a value of 0.01, resulting in a 99% probability to recover the key $k \in \{0, 1\}^Z$ if the given error is not above $HD = 8$-bit. The maximum tolerable error of $HD = 8$ bits was selected, as this represents a suitable trade-off between correctness and resource overheads. In addition, a register transfer level simulation using CHOICE PUF data from the robustness evaluation in Section 3.3.4 was used to get a cycle-accurate simulation and validate the results.

<table>
<thead>
<tr>
<th>Block</th>
<th>Num.</th>
<th>LUTs</th>
<th>FFs</th>
<th>BRAMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUF $Z = 128$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>abs.</td>
<td>256</td>
<td>128</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(%)</td>
<td>1.5</td>
<td>0.3</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Fuzzy Extractor</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>abs.</td>
<td>927</td>
<td>1,896</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>(%)</td>
<td>5.3</td>
<td>5.4</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Majority Voter $M = 15$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>abs.</td>
<td>192</td>
<td>544</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(%)</td>
<td>1.1</td>
<td>1.6</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>1,375</td>
<td>2,568</td>
<td>0.5</td>
</tr>
<tr>
<td>abs.</td>
<td>7.9</td>
<td>7.3</td>
<td>0.8</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.15 shows the resulting storage requirement in bytes for the helper data illustrated in orange (left y-axis) and the corresponding WCET in clock cycles (blue,
right y-axis) simulated in hardware for a range of different Hamming errors in bits. With voting over $M = 15$ measurements, it was possible to correct $\max_{T_a \in T_a} \{ \text{BER}_{T_a} \}$ on all boards except board $B1$ in no more than $6,402,866$ cycles ($= 64.0$ ms) when $1.1$ MB of helper data is provided. The $\max_{T_a \in T_a} \{ \text{BER}_{T_a} \}$ on board $B3$ is even correctable in max. $117,177$ cycles ($= 1.2$ ms) and with only $21$ kB of helper data. Note that for all correctable configurations, the best case is when the key is already available after only $281$ clock cycles since this is the time needed to process helper data of size $S_H (E = 1)$, representing the iteration of one element in the list of helper data. In conclusion, the resource and timing overheads caused by such PUF-based key generation are beneficial for the security of edge devices.

### 3.5 Summary

On edge devices manufactured from nano-scale transistors, Physical Unclonable Functions (PUFs) are a promising alternative to insecure permanent key storage since they convert natural process variations within the silicon of these devices into a device-intrinsic secret. However, the state-of-the-art practice for silicon PUFs targeting FPGAs is limited to fixed PUF designs that suffer from changing device conditions such as temperature and aging effects, as discussed in Section 3.1. This limitation is mainly due to the inability to adjust particular timing and delay paths within the silicon PUF once satisfactory statistical properties in terms of uniqueness, uniformity, and reliability are found.
As a remedy, this chapter presents, CHOICE a tunable FPGA-based PUF design that offers 6,144 different configuration levels to provide device-specific PUF responses. The concept of address-tunable propagation delays, introduced in Section 3.2, allows the probability of obtaining '1's in the PUF response to be increased or decreased. A series of experimental results on a group of six 28 nm FPGAs, presented in Section 3.3, demonstrated the effectiveness of the proposed approach in enhancing the uniqueness and reliability of the obtained PUF responses. In this way, CHOICE outperforms existing static PUFs not only in terms of resource utilization but also in terms of uniqueness, reliability, and temperature robustness over a wide range of configurations. In particular, it has been shown that CHOICE can be implemented in only a single FPGA slice while achieving very low Bit Error Rates (BERs) below 1.5% at normal temperature conditions and below 8% at extreme temperatures for the boards investigated in our experiments. In Section 3.4, it was demonstrated how cryptographic keys could be generated even under such extreme temperature conditions, leveraging CHOICE PUFs as the foundation of a device-intrinsic secret that witnesses both storage resource savings and security.

To conclude, due to the basic unclonability properties that make copying or stealing the PUF circuit useless, this approach has the advantage that the cryptographic key generated by the PUF is held only within the FPGA’s internal registers and only for the time that the FPGA is turned on. This increases the manipulation security considerably compared to a conventional key storage. Due to the generation via a one-way hash function, it is also impossible for an attacker to draw conclusions about the PUF response from the provided helper data, since it holds with any hash function, if the input changes by even a single bit, the entire hash value changes.
Booting Programmable SoCs Securely from Non-Volatile Memory

A typical feature of Programmable System-on-Chips (PSoCs) including an SRAM-based Field Programmable Gate Array (FPGA) is to load an initial hardware configuration into the FPGA along with a software boot process for the processor system after power-on. However, as discussed in Chapter 2, once deployed in the field, edge devices in general – and PSoCs in particular – are exposed to physical attacks against their hard- and software. In particular, tamper attacks targeting the manipulation of the initial hardware/software configuration, typically loaded from an external Non-Volatile Memory (NVM), are among the most severe threats that FPGA-based PSoCs can face [JHZ+17]. For example, an attacker could attempt to wiretap or intercept configuration data that must be stored and transferred on the NVM when the device is turned off. In addition to the configuration data itself, valuable Intellectual Property (IP), authentication and login credentials, or sensitive user or machine records often need to be held across power cycles on such persistent memory. Obviously, all this data makes the boot process a desirable target for attacks in a potentially untrusted environment.

As a countermeasure, PSoC vendors offer the option to initialize their devices in a secure boot mode. In this regard, earlier PSoC generations used the processor side as a slave device to the FPGA during the boot process. In contrast, Xilinx Zynq PSoCs, as used in this work, utilize the processor as the master for the boot process [Xil21]. Here, the processor loads the system configuration from the NVM while relying on the confidentiality of the keys stored within persistent key storage. As a result, the entire trust in the device depends on the privacy and integrity of these secret keys. For this reason, in particular, the secure storage and transfer of hardware/software configurations from NVM devices remain challenging in an untrusted environment. This problem becomes even more apparent when one considers that power-up is the only time at which a system can be assumed to be in a known unaltered, and hence secure, state.

Therefore, trust in an initial PSoC configuration can only be assumed if the NVM can be a) unambiguously identified, b) checked that an attacker has not modified the content, and c) en- and decrypted with cryptographic keys not stored off-chip but instead provided
in-chip, on-the-fly, i.e., during boot. For the latter, the PUF-based cryptographic key generation method previously presented in Chapter 3 could be used for the purpose of encryption and authentication. The presented overall approach is based on a novel, in-situ key provisioning followed by an isolated execution of security-critical cryptographic operations within the reconfigurable logic of the FPGA to guarantee a trustworthy compute platform, not only during boot but also during operational mode in an untrusted environment. Here, the FPGA’s reconfigurable logic serves as a hardware-based root of trust for authentication and integrity checking of encrypted data communicated between the PSoC and the NVM. Its central component is called Trusted Memory-Interface Unit (TMIU). The TMIU functions as an intermediate guard instance between the PSoC’s processor and the NVM that allows the validation of data transfers between these two entities right after power-on, without processor intervention. In addition, the TMIU enables the NVM content to be locked to a specific PSoC device without the need for an external key exchange. In this way, both entities merge to a permanent and immutable system for protecting proprietary IP and sensitive data. Consequently, a configuration can only be booted by the PSoC from NVM if it has been successfully authenticated and found to be unmodified. Moreover, this entanglement increases the trust and security level of the overall PSoC system and assures the system designer’s delivered IP is only usable in a previously authorized system.

The remainder of this chapter is organized as follows. In Section 4.1, an overview of the current practice to securely boot configuration data for hardware/software programmable SoCs is presented, followed by a discussion of related work. Then, Section 4.2 presents the steps and flow of a multi-stage secure boot process, leveraging the PUF-based cryptographic key generation. Subsequently, the structure and hardware design of the TMIU concept for secure NVM communication is presented in Section 4.3. Finally, in Section 4.4, the practicability of the proposed security schemes is demonstrated through their integration into the secure boot process of a Xilinx Zynq PSoC platform by evaluating the design objectives performance, power, and resource costs. A summary in Section 4.5 concludes the chapter.

4.1 Initial PSoC HW/SW Configuration

Due to volatile program and hardware configuration memories, both processor and FPGA of a PSoC need an initial device configuration after each system reset or reboot. For this purpose, NVM devices based on NAND cells such as either flash memory, embedded multi-media cards, or external SD cards serve as storage media. This type of storage is mainly used in portable and mobile devices where e.g., a solid-state drive (SSD) would require too much system area or power. However, all these NVM devices have in common that they combine highly integrated flash memory cells with a simple memory controller that relieves the processor from low-level flash memory management. In

---

\[\text{The contents of this chapter are based on and partly published in [SFB'20].} \]
particular, their vast storage capacity and high data bandwidth, combined with low power consumption, make them an ideal storage medium for data-intensive edge applications. These characteristics have led to their use in today’s embedded devices not only as data storage but also as NOR flash replacement for system boot up. As a result, such mass storage NVM devices have become the de facto standard, especially for hybrid hardware/software PSoC architectures to enable fast device configuration after reboot cycles.

The initial configuration of a PSoC loaded from NVM into main memory (DDR) (see also Section 2.1.1) is a process that relies on various hardware and software configuration files located on dedicated partitions on the NVM. Figure 4.1 illustrates such an NVM device structure that serves both as a boot medium for an initial hardware/software configuration and likewise as data storage capable to host an entire file system. A HW/SW configuration comprising of processor program code and FPGA bitstreams is loaded from the NVM into the device by first checking the information of the so-called Master Boot Record (MBR) on the first sector of the NVM (cf. green pattern in Figure 4.1). Thereby, the master boot record provides the information about the partition tables, which define attributes like number, type, and organization of the different partitions available on the NVM medium. To boot the PSoC, the MBR points to the address of a specifically formatted boot partition, indicated in Figure 4.1 by the blue patterns. The boot partition, contains all boot code required for software and hardware initialization. Xilinx PSoC devices request the boot code to be available on external NVM in the form of a boot image (blue hatches in Figure 4.1). The header of this boot image defines the NVM’s composition and contains, among other information, the addresses of the first stage boot loader (FSBL) and specifies whether a second stage boot loader (SSBL) or an FPGA bitstream or also both are present on the boot partition at the time of booting. Here, loading the bitstream into the FPGA’s internal configuration memory is necessary to configure the reconfigurable logic. For a correct processor setup, the second stage boot loader is required in a second step to load additional data, namely the device tree and the Operating System (OS) kernel into the main memory. On the boot partition, either the bitstream or the kernel accounts typically for the most considerable portion of memory storage. For instance, Figure 4.1 shows an FPGA bitstream of size 2 MB and an OS kernel of size 9 MB, as this corresponds to the configuration size required to setup the Xilinx Zynq xc7z010clg400-1 used in this thesis. The bitstream size depends heavily on the number of logic elements available in the reconfigurable structure of the chosen FPGA. For example, the largest device in the Xilinx Virtex FPGA family requires a bitstream size of 56 MB. The kernel size, in turn, depends on the configuration of the operating system and grows with the number of needed operating system functions. In fact, already the size of the FPGA bitstream together with the OS kernel highlight why NAND-based mass storage is chosen for PSoC devices.

While configuring the FPGA directly at boot is optional, support for operating system functionality is mandatory for edge devices in order to make use of the network capabilities of a full-fledged operating system, e.g., Linux, directly after boot. This operating
system setup requires the loading of a device tree that allows the kernel to finally use and manage the underlying hardware components. The kernel’s configuration can then boot the remaining parts of the operating system, such as device drivers, IP tables, and setup the remaining file system. In addition to the network capability, the file system is essential, in particular, for edge devices to enable the organization of large file and data structures and to make use of extensive software libraries. For this purpose, a separate partition on the NVM is typically dedicated to the file system. Such a partition is illustrated in Figure 4.1 by the orange grid, which can be – depending on the system application – several GB in size. However, while the file system is not required during boot, the data stored on this partition frequently contain sensitive information as well, e.g., login credentials or user and machine records. Obviously, such information requires equal protection.

Due to the large data size that must be stored on the NVM device, it is difficult to find suitable protection measures that do not impair system performance. Common
anti-tamper measures of PSoC vendors, as presented in Section 2.3.4, mainly focus on the integrity and confidentiality of data stored on the boot partition. To this end, bitstream encryption is supported by the vendors’ synthesis tools to prevent attackers from pirating a hardware design when stored in the field. Hard-wired crypto modules that allow decryption and authentication are integrated into the boot process of the PSoC to enable the secure initialization of the FPGA and operating system from the NVM.

In general, a secure boot process is always based on a root of trust from which the authentication of subsequent steps in the boot process takes place. In this way, it can be guaranteed that only previously authenticated data is loaded from the NVM into the main memory in each phase. Nevertheless, on Xilinx devices, the root of trust is based on the required decryption keys that are provided by key storage associated with these crypto modules on the device. Unfortunately, as already shown in Section 2.3.4, these key storage can be tampered with or even read out by physical attacks, which jeopardizes the entire secure boot process.

In addition to the risk of key theft, recent attacks have shown [EMP20; Bar20] that the authentication and verification of the entire NVM content is necessary to achieve an adequate protection level. For instance, if a secure boot mode does not authenticate the boot header or partition tables on the NVM, this data is vulnerable to malicious modifications. As a result, an attacker could modify the boot header and partition table addresses to jump to a specific address to load arbitrary code from there, bypassing all of the security measures intended by the vendor’s secure boot process. In addition to protecting cryptographic keys, it is also crucial to protect the contents of the file system partition with its data and information that must be stored and loaded from memory in the event of a power cycle.

Therefore, a trustworthy scheme for authenticating the NVM and its content are required to address these issues and ensure the confidentiality and integrity of the entire device configuration. In this regard, the TMIU proposed in this chapter interfaces the processor with the NVM for secure data communication. Moreover, the cryptographic operations required for this purpose are implemented and executed in hardware on the FPGA, thus not degrading system performance. As will be shown in the following sections, this offers several advantages over traditional software-only solutions and can also build on existing methods provided by the PSoC vendors. For example, changing security requirements over the product lifetime of the PSoC can be accommodated by adjustments to the cryptographic primitives used, while software vulnerabilities and performance or power constraints are kept to a minimum. Before presenting the concept and structure of the TMIU in detail, the following subsection presents related work on major attacks and countermeasures addressing the secure boot from NVM.
4.1.1 Related Work

In the recent past, several attacks have shown that the vendor-provided secure boot process for PSoCs can be bypassed. In addition to the physical, invasive and semi-invasive attacks targeting the physical key storage as introduced in Section 2.3.4, logical attacks were also published that expose vulnerabilities in the implementation of the vendor-provided secure boot mode. Given that PSoCs in general – and FPGAs in particular – are now an integral part of many security-critical systems, trust in these devices is severely undermined by such disclosures. For instance, in [KHP+19], security researchers published methods for direct FPGA bitstream manipulation for the Cisco Trust Anchor module (TAm), which uses a Xilinx Spartan-6 FPGA for hardware-assisted secure booting. The exploit described in [KHP+19] allows the attacker to bypass all trust anchor operations, including secure boot and the injection of persistent malicious implants into both the FPGA and an attached application processor. Since such trust anchor modules are present in a range of products like routers, network switches, and firewalls, the malicious injection can have severe consequences for worldwide network-driven data communication.

Although the vendor-provided secure boot option takes bitstream verification and encryption into account, the boot process cannot detect or even prevent tampering with the boot image metadata or the NVM’s partition tables, leaving this data vulnerable to malicious modifications, as shown subsequently. In [Bar20], security researchers revealed that the Xilinx Zynq UltraScale+ PSoC family’s “encrypt only” boot mode has design flaws that allow arbitrary code execution. In this case, the boot and partition header parsing was flawed because the boot header was left unauthenticated in the “encrypt only” secure boot on these PSoCs when the addresses specified therein are read. Consequently, an attacker who manipulates the addresses specified in the headers could alter the control flow of the boot process and execute malicious code. Since the vendor’s secure boot mode is deeply integrated into the PSoC silicon, this vulnerability can only be addressed by revising the entire silicon design.

Another recent attack is the Starbleed bug, which was discovered by Ender et al. and published in [EMP20]. This attack provides the possibility of a complete breach of confidentiality and authenticity on Xilinx Zynq PSoCs with the only condition that the JTAG interface is accessible on a fielded PSoC/FPGA device of the Xilinx 7-series. In this attack, an adversary uses access to the JTAG port in order to load a custom bitstream into the device and read the internal eFuse registers once the attack is launched. In addition, the attacker must be in possession of the encrypted bitstream, which he could obtain by, for example, eavesdropping on the bitstream as it is loaded from the external NVM during power-up. In doing so, the attack exploits two major design flaws. First, a boot register called the warm start address register is not cleared when the FPGA goes into a supposedly safe lockdown after tampering with the boot data. This is dangerous because usually, when a secure lockdown is triggered due to, e.g., insufficient authentication, an internally generated pulse should reset all configuration logic, including hard-wired
Initial PSoC HW/SW Configuration

The second flaw was in the Xilinx Zynq boot mode that decrypts the bitstream before checking its integrity. This means that an attacker who can manipulate the FPGA’s bitstream header can modify it to write one word of the encrypted bitstream to the warm start address register. Since the warm start address register is located after the FPGA’s hard-wired crypto modules from a data flow perspective and is precisely not directly cleared after a detected tamper, an attacker could read plaintext words of the bitstream from this register. Given the unsupervised environment where these devices are often operated, an attacker can now repeat this process until the entire bitstream is extracted in plaintext. Also, since the HMAC key on Xilinx PSoCs is likewise part of the encrypted bitstream, a valid HMAC digest could be spoofed with an arbitrary bitstream to break in addition, the configuration data’s authenticity. Coinciding with the publication of [EMP20], Xilinx plans to publish a design advisory that informs their customers of this vulnerability. The approach proposed in this chapter performs, as a remedy, full encryption and integrity verification of the entire NVM content, including the bitstream, the boot header, and the HMAC digest. Furthermore, this method does not use any vendor-specific secure boot mechanisms so that the approach described here is entirely self-contained.

The fact that the current generation of PSoC platforms uses the processor system as the master for the boot process [Xil21] creates new threats that can be exploited. In this context, the work of Jacob et al. in [JHZ+17] showed that direct manipulation of main memory during secure booting becomes possible by inserting malicious hardware into the bitstream. In particular, the processor-centric architecture of the current generation of PSoC becomes vulnerable if the loaded hardware design has been infiltrated with a hardware Trojan (cf. Section 2.2.1) that has direct access to specific addresses in main memory that may contain boot configuration data. In this case, the attack works as follows. At boot time, when the bitstream is loaded from the NVM to initialize the FPGA fabric, a malicious IP block (the Trojan) is also loaded into the FPGA’s reconfigurable logic. While the processor is still booting, e.g., to load the kernel with the second stage boot loader, the Trojan scans the main memory for the boot parameters. Once these parameters are found, the Trojan modifies them, e.g., such that the system continues to boot with an unauthorized kernel image from a server instance over the network, instead of continuing the boot process from the NVM. In general, it can be said that this form of attack is only feasible if the processor is already at a relatively advanced stage of the boot process since the required network capability of the processor must be present. Nevertheless, the approach proposed in this work addresses also such threats by specifying a hardware-centric secure boot process where the entire boot process is performed by direct authentication in reconfigurable logic instead of involving a processor.

One approach that also deals with the secure boot on PSoCs is the self-authenticating secure boot proposed by Pocklassery et al. in [PCS+18]. Similar to the approach presented in this chapter, a PUF is instantiated on the FPGA to provide the necessary decryption key for secure booting to avoid insecure permanent key storage. For this, an unencrypted
The authors of [PCS’18] state that their secure boot process is protected such that any tampering of the unencrypted PUF bitstream would result in a key regeneration failure. Possible shortcomings of this method are that the initial bitstream and the required first stage boot loader are stored unencrypted on a regular external NVM device, which is vulnerable to manipulation. Moreover, neither the deployed device can be locked to the intended configuration, nor is the configuration verified if the NVM memory is tampered with after a successful boot. Instead, the approach in this chapter is fundamentally different in that it employs the PUF to provide an inherent FPGA secret that is combined with a unique ID of the NVM during on-the-fly key generation to form an inseparable unit of device and configuration. In addition, the integrity of the transferred data is verified after secure boot in the proposed approach by computing a hash digest for each sector swapped between the processor and NVM. As will be shown, this solution takes into account the confidentiality and the integrity of the data during operation through hardware-based full memory encryption, including the boot header and all subsequent boot loaders. It is therefore not in the scope of this chapter to look at vulnerabilities of other peripheral resources used thereafter such as the main memory. For attacks targeting the confidentiality and integrity of main memory content countermeasures exists, see e.g., [GMD’16].

The following section presents the steps of the proposed hardware-centric secure boot process, together with the details of the PUF-based on-the-fly key generation.

### 4.2 Hardware-Centric Secure Boot Process

This section describes the multi-stage secure boot process for PSoCs from a fully encrypted mass storage NVM device and its system assumptions. Here, the boot process is assumed to start from a small, one-time programmable ROM (PROM) to provide the initial configuration of the Trusted Memory-Interface Unit (TMIU) after a power cycle of the PSoC. Thereby, the TMIU already contains a CHOICE PUF instance for secure key generation, as presented in Chapter 3. Accordingly, the PROM on the PSoC is already provided with the TMIU bitstream and a particular board-specific PUF configuration before shipment. In this way, the bitstream can be loaded into a designated FPGA area to serve as a trust anchor between the processor and the NVM.

The PSoC’s initial configuration is transferred to the NVM in encrypted form during enrollment (cf. Section 3.4.2) for secure boot in the field. This process is performed in a
4.2 Hardware-Centric Secure Boot Process

Figure 4.2: Flow diagram of the proposed hardware-centric 4-stage secure boot process from Non-Volatile Memory (NVM), including authentication and key generation steps (figure adapted from [SFB+20]).

secure environment after the initial PUF key generation, without the key leaving the device. Note, should the NVM be tampered with or even stolen in the field, its content remains confidential due to the prior encryption. However, any detected tampering will prevent the PSoC from booting – as desired. During the reconciliation phase in the field, the TMIU design then enables the isolated execution of key regeneration and other security-critical operations in the FPGA’s reconfigurable logic to ensure trusted data transfers, including configuration data and system files. Accordingly, the remainder of this section is organized as follows. Section 4.2.1 introduces the multi-stage secure boot process before presenting a scheme for on-the-fly key generation in Section 4.2.2, which builds on the PUF-based key generator from Chapter 3.

4.2.1 Boot Process

To strengthen the overall system security of a fielded PSoC edge device, the secure boot concept requires several successful validation steps to guarantee the integrity and authenticity of the NVM and its contents. The steps necessary to achieve such trustworthiness are shown in Figure 4.2, which illustrates the proposed 4-stage boot process to establish a hardware-based chain of trust to the NVM containing the boot image of the PSoC. In the following authentication scheme, each stage requests legitimization to pass control to the next consecutive stage to build the chain of trust. Thus, even if one of the stages is denied legitimacy, the system enters a secure lockdown mode to prevent malicious code execution. The verification steps of the individual stages of the boot process are explained in detail below.
In the first stage, after power-on, the TMIU is loaded from a PROM to a specific area of the FPGA’s reconfigurable logic. In this regard, Section 4.3 and Figure 4.3 provide the details of the TMIU architecture. In a first step, the PSoC is authenticated by the TMIU by reading its device DNA identifier $\text{ID}_{\text{SoC}}$ (cf. Section 2.3.4) from the PSoC’s internal eFuse register. If the identifier does not match a corresponding cryptographic hash, denoted as $\mu(\text{ID}_{\text{SoC}})$ in Figure 4.3 and present in the TMIU bitstream itself, the system already enters a secure lockdown mode. This measure tries to ensure that the bitstream is not loaded onto a different device. However, if successful authentication occurs, the communication between the NVM and the processor of the PSoC is initialized to establish an NVM communication channel.

After loading the TMIU, initializing it, and successfully authenticating the PSoC, the TMIU enters a memory identification mode (cf. Figure 4.2 stage 2) to establish access to the external NVM. In the following evaluation, the system assumes an SD card as a non-volatile mass storage device. Since it is everyday praxis in memory identification to authenticate a particular memory controller via a unique NVM identifier, the card is first requested by the TMIU to identify itself through its 128-bit card identifier (CID). This card identifier is factory encoded in the card’s internal registers and is, therefore, immutable. This identification is not SD card specific, as similar concepts exist for other NAND-based NVM devices such as flash or embedded multi-media cards. This card identifier, referred to as $\text{ID}_{\text{NVM}}$, is validated by the TMIU against a reference checksum also compiled into the TMIU bitstream to determine if the NVM’s data transfer functions can be securely activated. In stage three of the boot process, this public card identifier is then used as a public random seed for key generation to unlock the contents of the SD card. Consequently, the $\text{ID}_{\text{NVM}}$ does not match the expected checksum if a different NVM is used. In such a case, all I/O functions with the NVM are immediately blocked, and the entire PSoC is put into a secure lockdown.

The third stage, which is also fully implemented in hardware, involves PUF-based key generation followed by boot image decryption and authentication. Here, after generating the PUF output as presented in Chapter 3, the memory identifier $\text{ID}_{\text{NVM}}$ is used as a salt $\mathbf{s}$ together with the PUF’s response $\mathbf{r}_{b,c} \in \{0,1\}^Z$ as an FPGA-inherent secret, performing on-the-fly key generation, which is elaborated in more detail in Section 4.2.2. As discussed in Chapter 3, this guarantees the secure and reliable generation of a cryptographic key $\mathbf{k} \in \{0,1\}^Z$ of length $Z$ to decrypt the boot image and other data stored on the NVM. After decryption, additional authentication checks are performed. Since the entire NVM structure, including the boot partition, is analyzed, the encrypted master boot record is authenticated with partition tables and other metadata. Afterward, the integrity of the entire boot image data referred to as $\text{BI}$ is verified by computing the corresponding hash digest $\mu(\text{BI})$. For this purpose, the hash $\mu(\text{BI})$ is added to the boot image during the PSoC enrollment phase before it is stored encrypted on the NVM. During the reconciliation phase at boot and after successful key generation, the boot image’s integrity is verified by calculating the hash value within the TMIU while simultaneously loading the data from the NVM. The resulting hash is compared
Algorithm 1: PUF-based on-the-fly cryptographic key generation for $k \in \{0,1\}^Z$

1. \textbf{function} KeyGen($r'_b, S, M, C, ID_{NVM}$) // PUF resp., helper data, memory ID
2. \hspace{1em} $i \leftarrow 0$
3. \hspace{1em} $a \leftarrow 0$
4. \hspace{1em} \textbf{while} $i \neq E - 1$ \hspace{1em} // recover key before $i$ equals $E - 1$
5. \hspace{2em} $w'[i] \leftarrow M[i] \land r'_b \in \{0,1\}^Z$
6. \hspace{2em} $DK[i] \leftarrow PBKDF_{n,1}(w[i], S[i], Z + Y)$
7. \hspace{2em} $K \leftarrow C[i] \oplus DK[i]
8. \hspace{1em} \textbf{if} K[Z + Y : Z] = a \textbf{then} // extract key
9. \hspace{2.5em} $k \leftarrow \{0,1\}^Z \leftarrow PBKDF_{n,1}(K[Z - 1 : 0], ID_{NVM}, Z)$
10. \hspace{2.5em} \textbf{return} $k \in \{0,1\}^Z$ // return cryptographic key
11. \hspace{1em} \textbf{else}
12. \hspace{2em} $i \leftarrow i + 1$
13. \hspace{1em} \textbf{end}
14. \hspace{1em} \textbf{end}
15. \hspace{1em} \textbf{return} error // key could not be recovered

against the appended one when the entire boot image data is processed. In this way, any tampering of the encrypted boot image is detected by a different hash value, leading to an immediate abort of the boot process. During the boot process but also the regular operating mode of the system, the TMIU maintains the complete memory en-/decryption and hash calculation, e.g., when data is swapped to or from the main memory.

Finally, in stage four, the TMIU hands over control to the processor system on the PSoC to set up the remaining operating system and launch the user application if the boot partition has been successfully verified.

In conclusion, the system becomes operational only when stage one through four of the boot process shown in Figure 4.2 have been successfully validated and executed as expected. The following describes the on-the-fly key generation scheme and its implementation in detail.

4.2.2 On-the-Fly Key Provisioning

In the previously presented secure boot process, the key for decrypting the NVM content is provided quasi on-demand at boot time to mitigate the risk of key tampering. However, this is only the case if the previous authentication of the PSoC and the NVM device was successful, i.e., only if the combination of the PSoC identifier $ID_{SoC}$ and the NVM identifier $ID_{NVM}$ is accepted. This on-the-fly key generation is performed using the previously proposed CHOICE PUF with the fuzzy extractor introduced in Section 3.4.2 to enable secure cryptographic key expansion from the PUF.
The pseudocode in Algorithm 1 illustrates the procedure for recovering a cryptographic key of arbitrary length \( Z \) from a noisy PUF measurement \( r_{bc} \in \{0,1\}^Z \) for secure boot. In addition to the response of the PUF measurement \( r_{bc} \in \{0,1\}^Z \), the NVM device identifier \( ID_{NVM} \), and the set of helper data \( H \), consisting of the salts \( S \), the masks \( M \), and the ciphers \( C \) serve as input data for the key generation. Depending on their size, at least two possibilities arise for storing the public helper data. One option is integrating the helper data into the TMIU bitstream, e.g., synthesizing it directly into Block RAMs (BRAMs). Although this option is the fastest in terms of performance, since the helper data is instantly available in the FPGA, hardware memory resources are also costly, as the helper data can be extensive, as shown in Figure 3.15. A second option that does not require any extra FPGA resources would be if the helper data were available in an unencrypted portion of the NVM memory and the loading of this data from NVM for key generation occurred directly at boot time. Regardless of which of the two options is chosen, successful key generation depends on the number of bit errors in the PUF response, which, combined with the amount of helper data provided, determines whether or not the key can be reconstructed from a noisy PUF response.

Suppose that for key generation in the field, a number \( E \) of helper data entries are provided in the bitstream of the TMIU. According to the reusable fuzzy extractor, initially proposed by Canetti et al. in [CFP+16] and evaluated in Section 3.4.3, key generation is successful if the key is reconstructed before the entire list of helper data has been processed. In Algorithm 1 a key has been successfully recovered if the pre-defined extraction indicator \( \alpha \) is extractable from the extended key \( K \) shown in line eight of Algorithm 1. The algorithm starts with a zero-initialized loop variable \( i \). During each iteration, the noisy PUF response \( r_{bc} \in \{0,1\}^Z \) – which should be held in an internal hardware register after majority voting – is logical AND-connected with a provided mask \( M[i] \) as part of the helper data \( H \). The resulting value of this operation is then the FPGA-inherent noisy secret \( w[i] \), which is subsequently used as the secret input of a Password-Based Key Derivation Function (PBKDF) function along with the random salt entry \( S[i] \) providing an additional random input. In this way, the PBKDF uses an underlying HMAC algorithm based on a secure hash implementation \( \mu(D) \) embedded in the TMIU (cf. Figure 4.3) to derive the pseudorandom hash digest \( DK \) of length \( Z + Y \) bits. \( DK \) is XOR-linked with the corresponding cipher elements \( C[i] \) to derive the extended key \( K \) in line seven of Algorithm 1. Now, suppose the two most significant bytes of the extended key \( K \) equal \( \alpha \), which are defined as zero in Algorithm 1 (cf. line 3). In this case, the cryptographic key \( k \in \{0,1\}^Z \) is recovered by performing a final PBKDF computation (line nine in Algorithm 1), this time using only the NVM device identifier \( ID_{NVM} \) as random salt. If the key generator cannot recover the key with the provided list of helper data, key generation is terminated with an error in line 15. This termination causes the PSoC to go into a secure lockdown, as this could be an indicator that someone tampered with the PUF before booting. Finally, the result of Algorithm 1 is an FPGA-internal unique cryptographic key derived from the external
memory identifier $\text{ID}_{\text{NVM}}$ and a CHOICE PUF response that can be used as a symmetric de- and encryption key.

Concerning the security of this approach, the TMIU reads the unique NVM identifier $\text{ID}_{\text{NVM}}$ at the memory identification stage and compares this number to a reference checksum compiled into the TMIU bitstream (cf. Figure 4.2). So even if an attacker could read out the cards internal identification register or probes the communication signals, no plaintext attack on the key is feasible, because the attacker has no possibility to obtain the FPGA internal PUF response from a locked (i.e., JTAG-disabled) device. In addition, the keys are only applied within the programmable logic of the PSoC. Furthermore, a reset or power shutdown clears all generated keys and information. It follows that the card or any other connected memory device behaves similar to a passive dongle to tie permanently and immutable proprietary IP to an authorized device. Replacing either the memory device or changing its content or connecting a different PSoC device would result in a key generation failure and prevent the system from booting. Moreover, the NVM itself is only read- and writable in combination with the intended PSoC device. Nevertheless, for instance differential power analysis (DPA) side-channel attacks are still possible and require further protection measures such as key-rolling, where multiple keys are used. With this covered, the following section proposes the implementation of the TMIU within the hardware of the PSoC.

### 4.3 Trusted Memory-Interface Unit (TMIU)

This section describes the operation of a TMIU prototype and its building blocks. The developed prototype uses the Secure Digital Input/Output (SDIO) interface to access external non-volatile storage media such as SD cards and embedded multi-media cards. In this context, the following subsections introduce two custom IP cores developed to implement this interface. Section 4.3.1 presents a command control module that directly translates memory initialization and read/write commands replicating the communication between the processor and NVM. This command controller module also takes control over the multi-stage secure boot process proposed in Section 4.2, sending the PSoC into a secure lockdown if any of these stages could not be authenticated. The second module, presented in Section 4.3.2, is a data transport and verification module that de-, respectively, encrypts and verifies the data exchanged between the two entities memory and processing system without sacrificing performance. The last component of the TMIU is the key generation module, which comprises PUF, fuzzy extractor, helper data $\mathcal{H}$ and checksums to authenticate NVM as well as PSoC. Details of its operation and implementation have already been presented in Section 3.4.3 and Section 4.2.2 and are thus, not further elaborated here.
4.3.1 Communication Interface

An SD card was chosen to evaluate the proposed secure external memory communication scheme in Section 4.4. Therefore, a brief introduction to the SDIO protocol specification required for this purpose is first given to understand better the interaction between the TMIU and the SD card. The SDIO protocol is the standard not only for removable SD cards used in many consumer electronics devices such as cell phones, routers, and handheld computers, but also for its embedded counterpart, the embedded multimedia card, which is soldered on the board. The SDIO protocol is structured so that a host controller, in our case the processor of the PSoC, communicates with the memory controller as the slave via two dedicated interfaces according to the master/slave principle. Here, a data bus is used to read or write data from the host to the external memory on a bidirectional 4-bit parallel data interface. Because of the bidirectionality, this interface must carry both raw data \( D \) written to external memory by the processor and encrypted data \( E \) read from memory. In this process, data transfers to or from the SD card are performed as a single block read/write or in multiple blocks, depending on the transferred data size. A block transfer represents the data of a specific memory sector of 512 bytes from the card, which is always followed by an associated 16-bit polynomial value, calculated by a Cyclic Redundancy Check (CRC) function. This redundancy check is required to detect possible errors in the transmission. Also, this block-based communication enables sector-by-sector decryption, encryption, and hash calculation, despite random access to individual memory sections.

The actual data transfer is preceded by various controller-specific commands transmitted via a second bidirectional one-bit interface, denoted as (CMD) in the following. These commands control the execution of the communication protocol routines based on responses received from the NVM’s internal controller. Similar to the data bus, each of these command sequences are protected by a 7-bit CRC value to detect any bit flips that might occur during transmission. A re-transmission is triggered for the case that the calculated CRC values, whether for command or data transfers, do not match the appended value, adding an extra layer of safety to the protocol.

Since the TMIU is placed within the FPGA’s reconfigurable logic as an intermediate trust instance between the external non-volatile memory and the processor to monitor and control their communication behavior, a high-level description of the building blocks required for this purpose is provided in Figure 4.3. As can be seen, the ports for data communication – whether encrypted \( E \) or unencrypted \( D \)– and the command line (CMD), are the only access points of the TMIU to interface the processor and the attached NVM. In this way, the TMIU provides only the interfaces essential for memory transfer. As a result, no other potential user application running in parallel, either in software on the processor or in hardware on the FPGA, interferes with the TMIU. In this regard, Figure 4.3 on the left shows the ports connected towards external non-volatile memory. Here, the command controller component, referred to as NVM_CMD Module in Figure 4.3, observes the incoming commands sent and received by the NVM and the processor.
In this way, the command module can not only verify if the boot sequence proposed in Section 4.2 is followed but also take control over the communication between the processor and the NVM itself. More precisely, the NVM_CMD Module would immediately intervene and terminate the data transfer if any tampering is detected on either the storage or processor side.

As mentioned earlier, an installed SD card must authenticate itself via its card identifier (CID) during the memory identification stage at boot. Once the card sends this CID value, the NVM_CMD Module checks it and forwards it to the Key Generator Module. The Key Generator Module is in charge of checking whether the received memory identifier ID_{NVM} and the internally read device identifier ID_{SoC} match the pre-initialized checksums. In addition, the Key Generator Module reports to the NVM_CMD Module the status of cryptographic key generation and authentication by having access to the NVM_DATA Module. After successfully finishing the memory initialization and authentication step, the NVM_CMD Module permits the NVM_DATA Module the secure data transfer. Next, the NVM_DATA Module’s cryptographic building blocks that ensure confidentiality and integrity of the communicated data are presented.

### 4.3.2 Data Confidentiality and Integrity Verification

If not well designed, the TMIU with its cryptographic operations would have the potential to severely slow down not only the boot process but also the overall system performance. However, since high data throughput from the memory to the PSoC and vice versa is mandatory, the implementation of the NVM_DATA Module was designed with a primary focus on maximum speed. In particular, all internal cryptographic modules are interconnected via a streaming interface. This interface is designed for high data throughput and supports burst transmissions of unlimited size. Therefore, no addressing
mechanism or explicit synchronization is required, making it ideal for pipelined data streaming. Nevertheless, incoming data from the memory or the processor must first be proven non-faulty, e.g., against any transmission errors (i.e., bit flips). A CRC calculation does this for both encrypted data $E$ and unencrypted data $D$ before it is forwarded to a module that supports the required symmetric de- and encryption function denoted as $\epsilon (D, k_s)$. If a CRC check should be invalid, the data is sent to the PSoC unencrypted to trigger a CRC error resulting in a re-transmission. After decryption, resp., encryption, the CRC value must be calculated again and then appended to the data block before forwarding it to the PSoC/NVM.

The encryption module $\epsilon (D, k_s)$ supports both sector-wise de- and encryption and switches between these two modes depending on the communication direction. The same key that was used to encrypt the NVM content during the enrollment phase is now utilized in the field to write or read data to or from this NVM. In this way, it is achieved that only encrypted data is stored on the NVM at any time. Moreover, only in combination with the intended PSoC device, the NVM content is accessible.

For PBKDF implementation as well as integrity verification, the NVM_DATA Module utilizes a secure hash function $\mu (D)$. When the computation of the PBKDF for the cryptographic key generation is completed, the function $\mu (D)$ is used to calculate a boot image digest for comparison against the appended one $\mu (BI)$ after decryption. Suppose the calculated hash digest does not match the appended one. In that case, the NVM_DATA Module modifies the last byte of the image to provoke a CRC error on the processor side to invalidate the previous data transmission immediately. This invalidation is followed by the complete blocking of all further attempts to transfer data, which corresponds to the secure lockdown of stage three in Figure 4.2. Similarly, sector authentication is performed after successful boot when data is read or written to the NVM in operational mode. In this way, each sector is hashed, fully encrypted, and sent to the NVM making any manipulation immediately detectable. Later, when a reset or power off occurs, all internal registers are cleared, including the secret key, and the TMIU will again start the proposed multi-stage boot process.

Finally, the TMIU is intended to be separated from the user design, while the remaining non-configured area is left free and can be used for any FPGA user application. This is possible through partial reconfiguration and isolated place-and-route capabilities of modern FPGAs (see also Section 5.2.2). Specifically, partial reconfigurations of certain independent regions can be performed dynamically to update desired system functionality when deployed in the field. As a result, the TMIU, with all its crypto primitives, is logically and spatially isolated and runs completely independent of other user applications. In the following section, a concrete implementation of this TMIU prototype is presented, and a series of experiments are evaluated to measure its performance and power consumption.
4.4 Results

This section presents evaluation results of the proposed hardware-centric boot process and TMIU design. As for the PUF in Chapter 3, the Xilinx Zynq xc7z010clg400-1 PSoC placed on the Digilent Zybo evaluation board was again chosen for implementation. Although the implemented TMIU prototype was selected to support the SDIO protocol, this does not limit the proposed approach. Other forms of NVM with similar properties, such as solid-state drives or flash memory chips, would also be possible implementation candidates. The SD card used in this work complies with the “High Speed Mode” defined in the SD1.1 standard with a data transfer rate of 25 MB s$^{-1}$. To address the system assumptions introduced in Section 4.2, a one-time programmable ROM (PROM) is required to provide the bitstream of the TMIU at boot time. Since such a PROM is not available on the Zynq PSoC mentioned above, an on-board available programmable flash memory is used instead. To ultimately emulate the PROM behavior, this flash is connected via an SPI-flash controller and thus triggers the initial loading of the TMIU bitstream.

The bitstream size of the TMIU is taken into account to determine the throughput of the proposed design. Here, the memory requirement for the TMIU implementation stored on the PROM totals 1.9 MB. In addition to the PROM, a 16 GB SD card was used as non-volatile memory, directly connected to the pins of the PSoC’s FPGA. As shown earlier in Figure 4.1, the card memory was formatted with a 30 MB boot partition containing the PSoC boot image, device tree, and a Linux kernel image. The remaining card space was utilized as a Linux file partition to provide the root file system. In addition, the boot process evaluation considers the configuration of both processor software and hardware bitstream. For this reason, the boot image contains a partial FPGA configuration bitstream in combination with a second stage boot loader to load the kernel image and set up the Linux operating system on the device. As for the target clock frequency of the TMIU implementation in hardware, the clock speed dynamically adapts to the SDIO protocol during the memory initialization phase. For a “High Speed Mode” SD card, the maximum clock frequency of the SDIO protocol is 50 MHz. Last but not least, the Xilinx Vivado design suite 2020.3 was utilized to synthesize the TMIU on the Zynq PSoC mentioned above. Thereby, the design objectives performance, power, and resource costs were collected at a 20 ns clock cycle time in order to comply with the maximum clock frequency of the SDIO protocol.

4.4.1 Performance Evaluation

In the following, the boot time and the achievable data transfer rate between NVM and PSoC are evaluated using the proposed TMIU concept and implementation. Figure 4.4 shows the time required to load the TMIU configuration from the PROM in the first orange portion of the diagram. This time amounts to 98 ms for the initialization of the TMIU, mutual authentication, and key generation (steps 1-3 in Figure 4.2), which
corresponds to a throughput of 19.4 MB s⁻¹. Subsequently, the boot process continues from the SD card by loading and verifying the PSoC boot image and the remaining processor files. In this evaluation, data with a total size of 13 MB is loaded for a boot image with FPGA bitstream plus additional device tree and Linux kernel image. The time for this second boot is highlighted in blue in Figure 4.4. Here, the diagram shows that for an SD card specified with a maximum achievable line rate of 25 MB s⁻¹, a boot time of 526 ms can be measured, corresponding to a data rate of 24.7 MB s⁻¹. As can be seen, the TMIU design does not reduce the achievable throughput of the NVM device because it operates at the SD card’s maximum line, respectively, I/O rate. Therefore, the presented security measures do not degrade system performance even with extensive boot files since the throughput scales with the amount of data loaded during boot. Regarding latency, decrypting and authenticating data in hardware takes a total of 52 clock cycles to process a sector of size 512 bytes. Consequently, it can be said that the boot time is only limited by the bandwidth of the chosen NVM medium and its interface.

4.4.2 Hardware Resource and Power Overheads

The amount of FPGA resources required for the TMIU implementation in terms of FFs, LUTs, and 36K BRAMs – no DSP slices were required – is shown in Table 4.1. Table 4.1 lists the TMIU and its submodules for key generation, SDIO command control, and data processing. The hardware resource requirements for the key generation module have already been presented in Table 3.3. For this reason, in Table 4.1, the hardware resources for the helper data \( \mathcal{H} \) have been omitted. As described in Section 4.2.2, the helper data could come directly from the external NVM. Above that, the resources needed for the PBKDF, with the underlying hash function, are assigned to the NVM_DATA Module. As a
suitable hash function, the SHA-256 was deployed. For the de-/encryption function an AES-128 module in CBC mode was selected and accordingly a CHOICE PUF with a size of $Z = 128$-bit. Overall, the table shows that with 40% of LUTs and 18% of FFs on the

![Figure 4.5: Comparison of static and dynamic on-chip power on a Xilinx Zynq 7z010 PSoC separated by the PSoC’s processing system (PS) and FPGA logic, once without (a) and with (b) additional hardware resource overheads introduced by the TMIU security primitives.](image)

![Table 4.1: Resource requirements of the proposed TMIU implementation and its cryptographic submodules for secure boot on a Xilinx Zynq 7z010 PSoC [SFB+20].](table)

second smallest PSoC from the Zynq family, more than 60% of the hardware resources
would still be available for user applications which is tolerable concerning the increased security.

Regarding the power overhead caused by the additional security mechanisms, an estimate including both the overall design’s static and the dynamic power was obtained using the Vivado Power analysis tool. At the same time, switching activities were derived from timing constraints and simulation files. The results of this emulation are shown in Figure 4.5. Shown is a comparison of the on-chip power of the above Zynq PSoC device without the TMIU in Figure 4.5a and with the TMIU in Figure 4.5b. Here, the on-chip power is further divided into static power (blue) and dynamic power (yellow). Since more than 90% of the dynamic power is attributed to the PSoC’s processing system (PS7 in Figure 4.5), the dynamic power is further divided according to the individual FPGA logic elements that contribute to the dynamic power. As shown in Figure 4.5, when the additional FPGA resources introduced by the TMIU are taken into account, the total power consumption of the design increases from 1.67 to 1.77 W, which corresponds to an overhead of only 6%. For the total on-chip power with the TMIU, the processor consumes about 91% of the dynamic on-chip power. In comparison, the clock and register activity of the TMIU accounts for only 8% of the dynamic power, while only 7% of the total power consumption is attributable to static device power. Furthermore, resource, power, and timing overheads caused by the proposed TMIU approach are tolerable for the sake of security.

As a result, of all these evaluations it can be said that the lightweight TMIU implementation is applicable in quite small edge platforms. In addition, the proposed data module has been shown to achieve excellent performance numbers by operating at line rate.

4.5 Summary

Secure initialization of a PSoC-based edge device from persistent external memory is an indispensable necessity to ensure trustworthy operation in a potentially hostile environment. Therefore, to prevent any fraudulent replacement of the external memory or modification of its contents, this chapter presents a novel approach to verify the authenticity of the Non-Volatile Memory (NVM) and the integrity of the data stored on it.

Instead of boot ROMs, NVM devices such as SD and embedded multi-media cards or flash devices are typically used to store an initial hardware and software configuration of PSoCs due to their high capacity. The current practice to securely load hardware and software configuration data was discussed in detail in Section 4.1. Furthermore, in Section 4.2, particularly to mitigate the risk of key theft, a four-stage boot process was proposed. This boot process performs, in addition to the required verification and authentication steps, on-the-fly key generation at boot time to provide the decryption keys. Section 4.3 presents a fully hardware-centric security solution in which a so-called Trusted Memory-Interface Unit (TMIU) monitors the PSoC’s communication interface...
with the NVM device. Moreover, the TMIU orchestrates the proposed secure boot process, including authentication, integrity checking as well as, data de- and encryption.

Finally, the hardware implementation was evaluated in terms of resource utilization, on-chip power consumption, and performance. The experimental results presented in Section 4.4 demonstrate that the evaluated lightweight secure boot and TMIU implementation enhances the security of edge platforms and their data in untrusted environments without sacrificing system performance. In summary, the presented TMIU design is able to achieve data transfer rates at the I/O rate of the studied NVM device even with a limited number of computing resources. Additional security measures to prevent side-channel attacks such as analyzing power traces collected from the cryptographic operations performed inside the TMIU could be a direction of future research.
Updating Programmable SoCs in Untrusted Environments

Following the secure boot process, measures must be taken to maintain a device’s security throughout its lifetime in the field. This lifetime security can only be achieved if the device can be securely updated or upgraded even under untrusted site conditions, such as on a public network. Considering the long operating times of edge devices, e.g., for the industrial automation use case proposed in Section 2.1.2, this is particularly important. Furthermore, due to their adaptability of both software and hardware, this is especially true for PSoC-based edge platforms, as these are ideal computing platforms for products with a long lifetime. As a remedy, remote firmware updates have become the de facto standard to ensure the secure operation of such decentralized edge platforms.

However, as discussed in Section 2.2, the transfer and provision of such updates over an untrusted network are considered highly security-critical. A remote update must satisfy mandatory immunity assumptions, such as the authenticity of the update provider and the integrity and confidentiality of the content, which is typically loaded from an external cloud server. This chapter proposes a hardware-centric authentication scheme for remote PSoC updates and their contents to accomplish these security challenges. Specifically, an update protocol that relies entirely on tamper-proof hardware primitives is presented to enable secure reconfiguration of hardware and software functions of a PSoC-based edge platform from a remote cloud server. The central component, realized in reconfigurable logic, is called TUU. Employing a partial reconfiguration of the PSoC, the TUU is loaded for the time of an update to guarantee the integrity and authenticity of remote configurations. Apart from the fact that the partial reconfiguration flow causes only temporal hardware overheads, lightweight hardware modules are used to accelerate compute-intensive asymmetric elliptic curve operations even with a limited number of computing resources. In this way, the TUU enables device authentication, key generation, and key exchange mechanisms to be performed within the hardware without the key leaving the FPGA’s programmable logic. This isolation paves the way to provide a processor-free update validation and guarantees secret key confidentiality even in the event of the device’s software-based processing system being compromised.
Due to this strict separation of cryptographic operations and keys from the processor system, the risk of secret key or IP exposure is significantly reduced.

The remainder of this chapter is organized as follows: Section 5.1 outlines the update requirements for PSoCs before introducing existing related work. Section 5.2 presents the proposed update protocol along with the TUU. Section 5.3 demonstrated the effectiveness of the approach by evaluating performance, power, and hardware overheads. Finally, Section 5.4 briefly discusses the future prospects of the proposed protocol in terms of its resilience to quantum computers, followed by a summary of the chapter in Section 5.5.

5.1 PSoC Remote Configuration

Due to their deployment in the field of diverse applications and plants, PSoC-based edge platforms rely on support for remote updates in case of changing requirements or conditions. The deployment of these devices in the field, in particular, requires them to be controlled and programmed remotely to enable both software and hardware changes as needed, which raises new challenges regarding security. Especially after secure boot, a system should be upgradeable utilizing a remote update performed by a trusted authority. In this regard, when reprogramming during an update session, a) the source of the update (e.g., a Cloud Server (CS)) must be uniquely identifiable to protect the configuration of the device from malicious network sources, and b) the confidentiality of sensitive data such as secret keys and valuable IP must be ensured throughout the product lifetime.

As discussed in the previous chapter, such an update could replace an initially loaded boot image with a newer version in the NVM boot partition after an authenticated remote connection is established. In that case, an automatic reboot is required to load the new configuration. Alternatively, the old boot image could be kept as a backup if space permits and serve as a fallback if the new boot image fails to load as expected. Either way, updating keeps the entire edge device flexible and consequently reduces the total costs of ownership.

However, this flexibility can only be sustained if secure and trusted remote update mechanisms are provided that address the following security aspects. First, a suitable protocol for secure authentication and key distribution between server and edge device must be established. As discussed in Section 2.2.1, software-only solutions used for update protocols such as Secure Sockets Layer (SSL), Secure Shell (SSH), and Transport Layer Security (TLS) have often proven vulnerable. On the other hand, attempts implemented in hardware typically rely on parties sharing a symmetric key to reduce the hardware resource requirements. As a remedy, this chapter proposes an update protocol that solely leverages tamper-resistant hardware primitives to enable secure hardware/software reconfiguration of a PSoC from an external cloud server. Here, a hardware-centric asymmetric authentication of remote updates allows protection of the confidentiality

\[\text{The contents of this chapter are based on and partly published in [SWEP21].}\]
and integrity of the secret keys used while minimizing resource overhead through partial reconfiguration. Based on the focus on implementing the protocol in hardware, the following system security model can be formulated.

5.1.1 System Security Model

The following security model describes the general procedure of a remote update, considering the threats evaluated in Section 2.2, which are illustrated in Figure 5.1. A classical client-server principle is assumed, where one or multiple PSoCs, denoted in Figure 5.1 as PSoC_0, …, PSoC_{n-1}, act as clients and are connected to an external cloud instance representing the server (CS). This cloud server is part of a general, central device management system to remotely deploy arbitrary hardware/software configurations to the corresponding PSoC devices (cf. PSoC_2 in Figure 5.1). The update server is assumed to run on a trusted commodity. At the same time, the PSoCs operate in an untrusted environment and, therefore, are exposed to attacks. Such attacks could either be direct, as in the case of PSoC_1, or indirect by attacking the transfer and provision of an update, as in the case of PSoC_0. Specific implementation details of the data transfer interface, e.g., whether an update is transmitted by wire or over the air, are not considered in this context. Instead, a security model is formulated that dictates that the only way to reprogram the PSoC is to use a secure update session. That means JTAG and other programming interfaces are disabled and, in particular, the secret keys used by the PSoC are not accessible from its processing system at any time.

These system assumptions open up the attack vectors mentioned already in Section 2.2.1. For instance, the main goal of an attacker might be to eavesdrop, access, or modify proprietary and sensitive data transferred over a public network to each PSoC platform. This sensitive data includes IP in the form of object code intended to run on
the processor, full or partial hardware designs, or sensitive user data and keys. Another point that needs to be taken into account is that the update protocol should prevent attacks that target the availability of the PSoC. Here, denial-of-service attacks and man-in-the-middle attacks might allow an attacker to gain access to the connection medium and control the communication channel. They thus need also to be addressed. Since a sophisticated security model should also consider that an adversary could potentially compromise the software on the PSoC’s processing system in order to spy on secret keys, it can be formulated that these keys are neither accessible by the processor system nor stored on-/off-chip in memory or shared caches. Invasive attacks (cf. Section 2.2), including destructive methods to manipulate the FPGA as part of the PSoC to leak keys, are not considered. Finally, side-channel attacks such as analyzing power traces collected from the cryptographic operations performed inside the TUU are in the scope of future work. This chapter mainly addresses the integrity and confidentiality of the applied keys during an update by leveraging security-sensitive operations entirely within the programmable logic of the PSoC and, based on this, enable tamper-secure system reconfiguration. Therefore, the following subsection presents related work on protecting remote updates.

5.1.2 Related Work

Most remote firmware protocols for many edge devices, including PSoCs, are based on software libraries such as OpenSSL and the more recent TLS package [ZSA+19]. In the past, however, frequently hidden weaknesses in the implementation of these protocols have paved the way for cryptographic breaches with severe consequences. The most prominent in this context is the Heartbeat extension flaw known as Heartbleed [DLK+14]. Heartbleed was a simple buffer over-read bug in the OpenSSL library that allowed an attacker to read sensitive memory regions from affected devices. The attack allowed unauthorized memory access and thus access to sensitive data, including login credentials, cryptographic keys, and other private information. To address these software issues, solutions exist where the cryptographic primitives required for an update protocol are entirely implemented in hardware so that an attacker never gains direct access to the main processor or its shared memory instances such as DDR/RAM and caches, resulting in a tamper-proof isolation.

Update protocols based on hardware primitives have been driven in the past mainly by the industry’s need to protect valuable IP against unauthorized use, e.g., when updating FPGAs in the field. In this context, several protocols have been presented, of which those by Simpson/Schaumont [SS06], Guajardo et al. [GKS+07b], and Drimer/Kuhn [DK09] are particularly noteworthy. Unfortunately, all of these approaches rely on exchanging a secret key between a server instance and the FPGA. While such a symmetric key protocol provides the necessary confidentiality protection while keeping hardware resource requirements low, it also represents a single point of failure once the shared key is compromised. In addition, the remote server’s device management system (cf.
Figure 5.1 must store unique keys for all connected devices, resulting in burdensome device management. At the same time, it provides an attractive target for server attacks. As a remedy, this chapter proposes a public/private key exchange protocol that requires the server to have only its private key and the public keys of the connected devices.

While the general concept of such a public/private key protocol on FPGAs is not new, i.e., a possible implementation has already been described in [GKS+07b], more recent approaches have extended this to enable secure PSoC updates, e.g., Coughlin et al. in [CCW+19]. However, Coughlin et al. take the Linux OS software available on the PSoC processing system to generate the private/public key pairs, which would violate the restrictive trust assumptions formulated in Section 5.1.1. An attacker could potentially manipulate the software running on the processing system to break the processor’s memory protection barrier and compromise the confidentiality of secret keys. The hardware-based approach presented in this chapter differs by leveraging a CHOICE PUF circuit to generate a device-intrinsic key that never leaves the PSoC’s internal programmable logic, thus allowing the entire update process to be handled within isolated FPGA cells. At the same time, particular emphasis is placed on keeping the hardware resource overhead as low as possible. In particular, unlike [CCW+19], Elliptic Curve (EC) cryptography is used instead of RSA to reduce logic resources for key generation since smaller key sizes can be exploited in the process, as described in Section 2.3.2. With an additional partial reconfiguration flow, it becomes possible to reduce the hardware footprint further because parts of the logic are time-shared since the cryptographic building blocks are only needed during the update session. The details of the hardware-centric update protocol follow in the next section.

5.2 Secure Update Protocol

In Section 5.2.1, the remote update protocol for FPGA-based PSoCs from an external update server is described. The approach is based on the idea of an isolated authentication of the server identity, which is done by partial reconfiguration of the FPGA. The reconfiguration flow itself is described in Section 5.2.2. The TUU that provides the hardware building blocks to ensure trusted PSoC updates is detailed in Section 5.2.3.

5.2.1 Protocol

A schematic representation of the protocol exemplified by two communicating parties is shown in Figure 5.2. In this case, a Cloud Server (CS) (which runs on trusted commodity) and the reconfigurable logic of the FPGA as part of the PSoC representing the client (which operates in an untrusted environment, and hence, may be exposed to malicious modifications) are assumed. The steps required to transfer the desired update securely are proposed next.

In order to establish a trusted communication channel between the server and the
Figure 5.2: Sequence diagram illustrating the communication between the update provider (Cloud Server (CS)) and the FPGA as part of the Programmable System-on-Chip (PSoC) within the proposed update protocol. The cryptographic functions, keys, and system information used are also given [SWP+21].

PSocC, the cloud server must first have specific system parameters, such as information about the current firmware version of the PSoC. Besides the update, denoted by U in the following, the server needs its public/secret key pair, represented by $k_{CS}^U \in \{0, 1\}^Z$ and $k_{CS}^\mu \in \{0, 1\}^Z$. In the following, $Z$ always stands for the length of the respective key used. The concrete key length in bits used for the implementation is given in Section 5.3. The server also holds the public key of the FPGA $k_{FPGA}^p$ and the history of firmware versions $\nu$. A particular update version is defined as $\nu \in \nu$. On the other hand, the FPGA holds its public/secret key pair, denoted as $k_{FPGA}^p \in \{0, 1\}^Z$ and $k_{FPGA}^\mu \in \{0, 1\}^Z$. In addition, the FPGA has the public key of the corresponding cloud server $k_{CS}^U$ and the currently deployed firmware version $\nu$.

Before transferring the update, the server must send a Request Status to the FPGA. In this step, the server signs the version of the currently running PSoC firmware version $\nu$...
with its secret key $k_s^{CS}$. The resulting signature, denoted as $\sigma (v, k_s^{CS})$, is then sent to the FPGA to authenticate the server as trusted. The reason for requesting the firmware version of the PSoC is to prevent possible replay attacks, e.g., by malicious propagation of an old system configuration to downgrade the system to an older version. However, before processing an update request, the FPGA decides whether it is currently possible to load a Trusted Update Unit (TUU) (see Figure 5.4) by partially reconfiguring the FPGA\(^2\). In addition, the integrity of the TUU is verified using a pre-computed hash value labeled as $\mu$ (TUU), which is hard-coded in a specific FPGA area that is considered static. As discussed in more detail in Section 5.2.2, this enables a secure sharing of hardware resources and seamless updates without compromising system availability.

Once the TUU has been loaded and its integrity verified, the authenticity of the cloud server is proven using the sender’s public key $k_{CS}^{P}$. This key is hard-coded in the hardware design of the TUU. Alternatively, the key can be stored on external NVM to save hardware resources. If its signature cannot identify the CS, the update process is immediately interrupted to prevent possible denial-of-service attacks. Upon successful authentication, an asymmetric key pair is generated for the update session, denoted by $k_{s^{sess}} \in \{0, 1\}^Z$ and $k_{p^{sess}}$, respectively. Furthermore, the FPGA can now utilize the secret session key $k_{s^{sess}}$, and the cloud server public key $k_{CS}^{P}$ to compute an ephemeral key denoted $k_{eph}^{s}\in \{0, 1\}^Z$ for a symmetrically encrypted update channel. This ephemeral key is a secret, symmetric key that reduces the de- and encryption overhead for any received/sent update data compared to a fully asymmetric approach.

After calculating the ephemeral key on the FPGA, the public update session key $k_{p^{sess}}$ is sent to the server in a Respond Status step. This message is not encrypted since the public update key doesn’t have to be secret. Nevertheless, the FPGA signs the public session key $k_{p^{sess}}$ with its private key $k_{FPGA}^{S}$ to confirm authorship. Now, the cloud server can verify if the signed public update session key $k_{p^{sess}}^{s}$ originates from the intended PSoC device. If the verification is successful, the remote server can securely agree on the ephemeral session key $k_{eph}^{s}$ by using its private key $k_{CS}^{S}$ and the sent public session key $k_{p^{sess}}^{s}$ to calculate the symmetric ephemeral key for update encryption. These steps form the basis of the well-known Diffie-Hellman algorithm performed in our work by the TUU through an Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) key exchange. As a result, the remote server can encrypt the PSoC update using this ephemeral key to protect confidentiality when sending the data over an untrusted network. Moreover, by generating unique session keys for every update session, the protocol achieves Perfect Forward Secrecy (PFS) since an attacker cannot decrypt past communication. This forward secrecy holds even if the secret session key $k_{s^{sess}}^{s}$ gets compromised since such a key is always used for only one update. In this way, at least past update messages are protected if the FPGA’s secret key $k_{FPGA}^{S}$ is compromised. However, if this were

---

\(^2\)The exact procedure as well as the prerequisites of such a partial reconfiguration will be described in Section 5.2.2.
to happen, it would be possible to forge the authorship of the FPGA, thus possibly compromising all future messages.

As soon as the secure channel is established, the cloud server can send the symmetrically encrypted update data, denoted as $\varepsilon$ with the ephemeral key $k^{eph}$, in the Send Update step. In addition to the update itself, this message contains information about the new firmware version $v$ to update the current version parameter and, if necessary, a new public key of the cloud server $k^{CS}_p$. The communication of a new public key offers the possibility to revoke the server’s private key. This revocation has the advantage that a secure key of the cloud server $k^{CS}_c$ can be rejected if it is outdated, disclosed, or if employees knowing the key leave the company. In this context, however, it is necessary to note that this revocation does not apply in the event that the server itself is compromised. For instance, if an attacker has full access to the cloud server, he or she could compromise the cloud server’s private key $k^{CS}_s$. After manipulation, the attacker could apply her/his own (malicious) key called $k^{CS_y}_c$. Consequently, with this malicious key, the server can update the client using a malicious server public key $k^{CS_y}_p$ to gain control of the client. Suppose the attacker can send a malicious update $U^+$ to the client before a legitimate server can revoke the malicious key $k^{CS}_c$. In that case, the attacker gains full access as the server’s public key is replaced on the client as part of the update that depends on the previous (now compromised) public key $k^{CS}_p$. However, in this work, the central cloud server is considered secure because, unlike the client, the server is usually embedded in a well-protected corporate network. This work is aware of this attack scenario. Yet, the practicality of the proposed protocol is explicitly prioritized at this point, which is done according to the system and attack model formulated in Section 5.1.1.

When it comes to the integrity of the update, the server appends a secure hash value $\mu (U)$ to the message, which represents the content of the update in an unforgeable and compressed form. In this way, the FPGA can prove the integrity and detect if the update has been tampered with by calculating its own secure hash value over the received update data in the same way. Consequently, if an attacker tampered with the update data, the hash $\mu (U)$ calculated by the FPGA would not match the one sent. In this case, the update procedure could be halted for a specific time, and it would be possible to send a report about the incident to the cloud server.

After decrypting the received hardware/software update on the FPGA side, the update can be loaded directly into a system DDR or persistently stored on the NVM. While the first option is only a temporary update, lost after powering off the system, the second option requires a secure boot cycle of the PSoC, as suggested in previous Chapter 4, to be finally deployed. The last step is to send the Confirm Update message from the FPGA to the cloud server to confirm that the update has been completed and the firmware version has been updated. This completion happens once the FPGA sends the signature of the updated firmware version $v$ back to the server.

Since all keys must be only accessible within the FPGA and remain secure throughout
the product lifetime, the required key generation is based on a PUF and a RO circuit, as discussed in more detail in Section 5.2.3. In the following, a method is presented that protects these keys while still keeping hardware resources low.

### 5.2.2 Partial Reconfiguration and System Availability

This subchapter describes the partial reconfiguration of the FPGA to time-share the logic resources required for an update session and explains how the formulated security requirements are met. Therefore, the FPGA’s programmable logic was divided into two sections to enable seamless system updates, as shown in Figure 5.3. Here, Figure 5.3 illustrates the FPGA’s reconfigurable area, consisting of a static area (not reconfigurable at runtime) in black and a dynamic area, shown in light red. It holds while the dynamic area can be reconfigured at runtime, the static area of the FPGA always remains unchanged. Furthermore, the design of the static area consists of a configuration controller, which has direct access to both the DDR and the NVM of the PSoC-based edge platform and can also write bitstreams directly into the configuration memory of the FPGA. This direct write to the configuration memory is done through a dedicated interface called the Internal Configuration Access Port (ICAP) on Xilinx PSoCs. It is important to note that the ICAP interface does not limit the proposed approach to Xilinx-based FPGAs since, e.g., Intel/Altera FPGAs support a similar interface in the form of a dedicated IP block called Partial Reconfiguration IP (PR-IP) to send partial bitstream data directly to the FPGA configuration memory.

However, to avoid impairing system availability, the controller decides whether an update request can be serviced instantly, as this would trigger a partial reconfiguration of the FPGA to load the TUU into the dynamic area (indicated by the orange cells in the upper layer of Figure 5.3). With this default “delayed update”, the update procedure, and thus reconfiguration, can only begin when no safety-critical applications are running or when the allocation of other user applications (indicated by the blue cells in Figure 5.3) allows it. This strategy protects the PSoC from denial-of-service attacks, where a potential attacker controlling the network communication channel could otherwise make endless update requests. After verification, the TUU bitstream is forwarded to a SHA module in the static area. The corresponding hash is compared with a precomputed signature hard-coded in the configuration controller to perform an integrity check. Only if this verification is successful, the reset of the TUU is released, and the update protocol is processed further. Moreover, this way, the FPGA side has complete control over the entire protocol processing. Furthermore, only the FPGA can reprogram the PSoC using a secure update by writing the contents either to the NVM, DDR or in the case of a hardware-only configuration, directly to the FPGA’s configuration memory. Thus, an attacker with access to the external DDR of the processor system could attack the availability by tampering with the storage of the TUU bitstream, but could never compromise the confidentiality and integrity of the secret keys.

Regarding the security of the static area after power-on, it is assumed that the bit-
Figure 5.3: Loading the Trusted Update Unit (TUU) (orange cells) by partially reconfiguring a dynamic area of the FPGA (red cells) during a secure update session (upper layer). A specific “always available” area of the FPGA (black cells) is required to verify the integrity of the TUU itself. Partial reconfiguration only introduces a temporary allocation of hardware resources, while the availability of e.g., user applications (blue cells) is guaranteed.

Stream is protected against tampering attacks by a secure boot process, as presented in Chapter 4. The bitstream containing the TUU does not need to be encrypted since it does not have any direct secrets (i.e., no keys). It thus can be loaded directly from DDR into the designated area of the FPGA’s reconfigurable logic. In summary, the separation into a static and a dynamic area not only reduces the hardware footprint of the proposed approach but also enables seamless system updates while meeting the security requirements defined in Section 5.1.1.

5.2.3 Trusted Update Unit (TUU)

This subsection presents the necessary building blocks for the secure implementation and processing of the proposed update protocol, as it was shown in Figure 5.2. The following cryptographic algorithms have been implemented in dedicated hardware blocks for proof of concept. For this purpose, first, the FPGA’s private key is provided within the TUU hardware via a $Z = 163$-bit PUF as an extension of the design proposed in Chapter 3. Furthermore, to generate a new random session key for each update, the protocol requires, in addition, a True Random Number Generator (TRNG). In this context, hardware-based random number generators, especially free-running ROs, have proven to be an inherent source of entropy for generating unpredictable true random numbers. The underlying concept of an RO for random number generation is similar to the idea for generating PUF responses presented in Section 3.1.4, as it also uses the
oscillator’s phase jitter. The concatenation of multiple of these ROs forms a natural entropy source for a TRNG, allowing for the generation of an unpredictable, tamper-proof random number for each update session. For the proposed update protocol, a 32-bit ring oscillator-based TRNG, as presented in [BLM+11], is used to provide the private session key $k_{s}$ at runtime. For a detailed description of RO-based TRNGs, the reader is referred to reference [BLM+11] at this point.

Several cryptographic modules provided by TUU rely on the PUF- and RO-based key provisioning. Figure 5.4 provides an overview of these modules and their interaction with each step of the update protocol presented in Section 5.2. An Elliptic Curve Digital Signature Algorithm (ECDSA) based on the Koblitz sect163k1 curve supports asymmetric message signing and verification operations. According to the foundations of ECC introduced in Section 2.3.2, the curve is defined over the finite field $\mathbb{F}_{2^{163}}$ with $a, b \in \{0, 1\}$ and the reduction polynomial $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$. The Koblitz sect163k1 curve requires the PUF to provide a $Z = 163$-bit response after loading the TUU from which the secret device key $k_{FPGA}$ is derived using the fuzzy extractor-based key generation module presented in Section 3.4.2.

After successful PUF-based key generation, the key is passed to an EC Key Creation module where the corresponding public key $k_{p}$ is subsequently obtained by elliptic
curve point multiplication (cf. Section 2.3.2). Now, when the EC Verification module verifies the signature $\sigma(v, k^{CS}_p)$ of the cloud server in the Request Status step (output = 1), the EC Key Creation is invoked to generate the public session key $k^{sess}_p$ from the private session key $k^{sess}_s$.

When creating the signatures for the Respond Status and Confirm Update messages, the EC Signature module is responsible for signing the corresponding messages with the FPGA private key $k^{FPGA}_s$. Moreover, to compute the ephemeral symmetric key $k^{eph}$, the session secret key $k^{sess}_s$ and the cloud server public key $k^{CS}_p$ are processed by a module implementing the EC Diffie-Hellman algorithm. In the next step, when the cloud server sends the Send Update message, an AES decryption block applies the ephemeral key $k^{eph}$ to decrypt the transmitted update data. The received update data is decrypted using a 128-bit AES decryption block. Then, the integrity is checked by a 256-bit SHA module as part of the static area not shown in Figure 5.4. Finally, the configuration controller is responsible for writing the received update data either temporarily to DDR or via ICAP to the FPGA’s configuration memory or permanently to the NVM. After the update has been applied, the current firmware version $v$ is signed by the EC Signature module, and only the signature is sent to the cloud server as Confirm Update status.

During the entire update process, the software on the processor system only forwards the protocol commands between the cloud server and the TUU on the FPGA side. Thus, once an update is triggered, the FPGA handles all computationally intensive and security-critical operations. At the same time, the PSoC’s processing system is only responsible for communicating the sent/received messages to/from the cloud server. As a result, key generation, message signing, and verification can be performed without passing the secret keys to the processing system, thus remaining protected even in an infiltrated processing system. Hardware resources, performance, and power overheads of this method are presented in the following section.

5.3 Results

In this section, the evaluation results of the proposed hardware-centric update protocol and TUU design are presented. Due to the higher complexity of the asymmetric cryptographic modules, instead of the Xilinx Zynq xc7z010clg400-1, the slightly larger Zynq xc7z010clg400-1 PSoC found on the Avnet ZedBoard evaluation platform is chosen for implementation in this chapter. Besides the security of the TUU, the update process itself leveraging the partial reconfiguration of the FPGA is highly time-critical in order to impair not with system availability. In combination with the performance characteristics, the amount of hardware resource requirements introduced through the proposed security primitives determines the practicability of the update scheme. Therefore, and similar to the previous chapters, the Xilinx Vivado design suite 2020.3 was used to synthesize the TUU on the Zynq PSoC platform. Subsequently, the synthesized design was additionally investigated in terms of performance and power consumption.
for a target clock frequency of 150 MHz. Results and findings of this investigation are presented in the following.

### 5.3.1 Performance Evaluation

Since a lengthy update procedure might not be acceptable for some edge applications, the first design goal was to investigate the end-to-end latency of the steps required to perform the update protocol. For the update of the PSoC mentioned above, a firmware image of size 4 MB was chosen and sent over a wired connection from a host machine emulating the cloud server. As the transfer latency of the update depends on the transmission interface, this time is not considered. Therefore, only the measured latencies and absolute times for the individual steps performed within the PSoC are shown in Table 5.1. The TUU in the dynamic area of the FPGA processed each verification and processing step at the mentioned clock frequency of 150 MHz.

Table 5.1 shows that the latency to reproduce the secret $Z = 163$-bit FPGA authentication key from a noisy PUF response is 849 clock cycles. In contrast, the time required to generate the asymmetric session key pairs is more computationally intensive. Indeed, it takes almost as long as computing the shared ephemeral key using the Diffie-Hellman algorithm. This similarity in time is because a single elliptic curve point multiplication is performed on both operations. The additional six clock cycles to generate the session keys are caused by the TRNG, which generates 32 random bits per clock cycle and therefore requires six clock cycles to generate the required key length of 163 bits. In addition, the time needed to verify a signature is higher than that needed to create a signature, while the slight discrepancy in clock cycles for `Respond Status` signing and `Confirm Update` signing is due to slightly different input values. Finally, most of the time is spent decrypting the update. However, since this time is linearly dependent on the update size, this is thus only determined by the throughput of the AES module. Compared to the time for update checking and decryption, loading the TUU bitstream

<table>
<thead>
<tr>
<th>Protocol Step</th>
<th>Latency in cycles</th>
<th>Latency in [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Key Generation</td>
<td>849</td>
<td>0.006</td>
</tr>
<tr>
<td>Verify Request Status</td>
<td>363,448</td>
<td>2.423</td>
</tr>
<tr>
<td>Gen. Asym. Sess. Key</td>
<td>138,098</td>
<td>0.921</td>
</tr>
<tr>
<td>Sign Respond Status</td>
<td>240,389</td>
<td>1.603</td>
</tr>
<tr>
<td>Diffie-Hellman Algorithm</td>
<td>138,092</td>
<td>0.921</td>
</tr>
<tr>
<td>Decrypt Update</td>
<td>3,792,826</td>
<td>25.285</td>
</tr>
<tr>
<td>Sign Confirm Update</td>
<td>232,751</td>
<td>1.552</td>
</tr>
</tbody>
</table>
by partial reconfiguration is negligible with a measured 436 ns.

### 5.3.2 Hardware Resource and Power Evaluation

In terms of power consumption, an estimate of the static and dynamic power of the overall design, including the TUU netlist, was carried out using the Vivado Power analysis tool similar to the TMIU investigation in Chapter 4. The results of this estimation are shown in Figure 5.5. Compared is the on-chip power of the 7z020 Zynq PSoC device without TUU in Figure 5.5a and with TUU in Figure 5.5b. In addition, the on-chip power is further divided into static power (blue) and dynamic power (yellow). The measurements showed an increase from 1.67 to 2.04 W for the total power consumption of the design, which corresponds to power overheads of 18%. The numbers reveal that the processor contributes to about 79% of the on-chip dynamic power within this total power. In comparison, the TUU’s clock and register activity are responsible for just 21% of the dynamic power. The share of static device power in the overall power consumption is 8%, thus tolerable for often stationary operated edge devices.

Finally, as a third design objective, special attention was paid to area efficiency since the resource overhead of the update unit should not be larger than an actual user application. Therefore, Table 5.2 provides the FPGA resources required for a concrete protocol implementation, divided into its static and dynamic FPGA portions (cf. Figure 5.3). Here, the static area represents the blocks that are permanently not usable
for user applications and comprises the logic of the configuration controller, including the Internal Configuration Access Port (ICAP) interface and the SHA-256 module to configure and authenticate the TUU. In comparison, the TUU contains, as part of the dynamic area, the PUF key generation module including the fuzzy extractor (which implements the PUF-based key generation scheme introduced in Chapter 3). Further components implemented include a TRNG used for ephemeral key generation, an EC key and signature creation module, necessary for the Elliptic Curve Digital Signature Algorithm (ECDSA) and the Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) key exchange algorithm, and an AES decryption module for 128-bit data decryption.

The numbers in Table 5.2 demonstrate that with 39% LUTs and only 15% FFs, more than 60% and 80% of the available LUTs and FFs on the 7z020 Zynq, respectively, are at the disposal of the user to design custom hardware applications that never interfere with the TUU or the update process. This means that more than half of the resources of the FPGA can be used by the user for edge processing, while 37% of the FPGA’s LUTs logic must be released to load the TUU when an update is required. Overall, based on these figures, it can be concluded that the resource, power, and performance overheads caused by the proposed TUU approach are tolerable in the interest of tamper-proof hardware security for edge platforms.

5.4 Perspective Post-Quantum Security

Cryptographic schemes and key lengths considered secure today could become insecure within the next decade. This insecurity arises from rapid developments in quantum computing, which leads to the fact that established cryptographic methods – including those presented in this thesis – will sooner or later have to be replaced by so-called Post-Quantum Cryptography (PQC). This awareness can be traced back to two quantum algorithms, namely Shor’s [Sho97] and Grover’s [Gro96] algorithm, since they break the computational security guarantee that holds for today’s computing power.

The Elliptic Curve (EC)-based asymmetric cryptography applied to the secure update protocol in this chapter relies on the hardness of finding discrete logarithms on traditional computers, as introduced in the fundamentals in Section 2.3.2. However, Shor’s algorithm [Sho97] finds discrete logarithms in polynomial time on a hypothetical quantum computer, leaving cryptosystems based on such elliptic curves insecure. The same polynomial execution applies to public-key cryptosystems using integer factorization (cf. Section 2.3.2). Given this fact, classical public/private key systems can no longer be protected by increasing the key length and will have to be replaced as soon as “large enough” quantum computers become a reality.

In symmetric-key cryptosystems, on the other hand, the consequences of quantum computers are considered manageable from both a crypto theoretical and technical point of view. Particularly, Grover’s algorithm [Gro96] has shown that the complexity

\[3\text{Large at this point refers to the number of stable Qbits that are required for reliable quantum computing.}\]
to finding a symmetric key for a given plain- and ciphertext is reduced from $O \left( 2^Z \right)$ on a traditional computer to $O \left( \sqrt{2^Z} \right)$ on a hypothetical quantum computer, where $Z$ denotes the key length in bits. From this reduction in computational complexity, one can deduce that doubling the size $Z$ of the symmetric key $k_s \in \{0, 1\}^Z$ is sufficient to restore security in symmetric cryptosystems, such as the one discussed in Section 2.3.1. In this work, this key size doubling would concern all AES and HMAC operations used in the context of PUF-based key generation, secure boot, and update.

Due to this vulnerability of classical cryptosystems, the National Institute of Standards and Technology (NIST) has initiated a standardization process to establish alternative public-key cryptosystems that resist attacks enabled by quantum computers [Nat16]. NIST still evaluates these PQC algorithms at the time of writing regarding security, performance, and practicability. Given the uncertainty of which of these algorithms will be secure for the coming decades and taking into account the challenges that arise in the transition towards establishing a new cryptographic infrastructure, special attention should be paid to platforms that support crypto-agility necessary for this purpose [WAG+21]. In particular, edge platforms with a long lifetime could offer existing

<table>
<thead>
<tr>
<th>Module</th>
<th>Block</th>
<th>Num.</th>
<th>LUTs</th>
<th>FFs</th>
<th>BRAMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Area</td>
<td>Conf. Controller</td>
<td>abs.</td>
<td>106</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>(%)</td>
<td>0.2</td>
<td>0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SHA-256</td>
<td>abs.</td>
<td>1,217</td>
<td>1,358</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>(%)</td>
<td>2.3</td>
<td>1.3</td>
<td></td>
<td>0.4</td>
</tr>
<tr>
<td>Dynamic Area</td>
<td>PUF ($Z = 163$)</td>
<td>abs.</td>
<td>326</td>
<td>163</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>(%)</td>
<td>0.6</td>
<td>0.2</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Fuzzy Extractor</td>
<td>abs.</td>
<td>927</td>
<td>1,896</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>(%)</td>
<td>1.7</td>
<td>1.8</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>TRNG-32</td>
<td>abs.</td>
<td>192</td>
<td>544</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>(%)</td>
<td>0.4</td>
<td>0.5</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>AES-128</td>
<td>abs.</td>
<td>1,515</td>
<td>1,054</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>(%)</td>
<td>2.9</td>
<td>1.0</td>
<td>2.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EC (sect163k)</td>
<td>abs.</td>
<td>16,454</td>
<td>11,472</td>
<td>9.5</td>
</tr>
<tr>
<td></td>
<td>(%)</td>
<td>30.9</td>
<td>10.8</td>
<td>6.8</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>abs.</td>
<td>20,737</td>
<td>16,505</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(%)</td>
<td>39.0</td>
<td>15.5</td>
<td>10.0</td>
<td></td>
</tr>
</tbody>
</table>
hardware components to be reprogrammed or even a entire replacement of security algorithms if, for example, the proposed FPGA-based PSoC devices are deployed. Future work could thus address the reevaluation and extension of the cryptographic schemes presented in this work to resist attacks enabled by quantum computing.

5.5 Summary

This chapter introduced solutions for a hardware-centric update procedure to verify the authenticity of a remote update provider and the integrity of transferred configuration data for FPGA-based edge platforms. This protocol aims to guarantee the protection of cryptographic keys even in the case of an infiltrated processor system.

Therefore, in the first section of this chapter, the challenges of secure remote updates were presented. Based on this, a security model was formulated that addresses both the risk of key theft and the manipulation of sensitive configuration data. Subsequently, Section 5.2 presented the specific steps of the update procedure, which are performed entirely within the hardware of the PSoC’s FPGA to prevent tampering with configuration data provided and transmitted by a remote update server. In addition to the protocol itself, the partial reconfiguration flow of a dedicated hardware design called Trusted Update Unit (TUU) is described, which is responsible for initializing the PSoC’s communication interface with the server, including authentication, integrity checking, and decryption of data. This update unit avoids the need for external key storage by generating ephemeral keys within the FPGA’s programmable logic to achieve Perfect Forward Secrecy (PFS) of the protocol, which prevents an attacker from decrypting past communication should symmetric communication keys be compromised. The protocol and its hardware implementation were evaluated in detail in Section 5.3 concerning security, resource utilization, and performance. Thanks to the partial reconfiguration of large parts of the protocol implementation, the TUU can already be used in reasonably low-cost, even entry-level PSoC devices. The chapter concludes with a brief discussion on how the proposed security protocol can be extended to provide protection given the possible emergence of quantum computers in the near future.
Summary and Future Directions

A wide variety of internet-enabled devices in domains such as autonomous driving, remote health, and industrial automation have led to exponential growth in digital data. As the capacities of the network infrastructure could noticeably no longer keep up with the sheer volume of this data, a clear trend towards data processing at the edge of the network has been seen in recent years. As a result, more and more computationally and thus resource-intensive processing tasks have been shifted to the data source, i.e., the sensors. Today, both industry and academia are debating what future computer architectures should look like to meet the demands of near-sensor data processing, which often must be performed under strict real-time conditions. The consensus is that scalable and thus flexible platforms are needed to handle such data-intensive and time-critical tasks right at the network’s edge. To that end, this thesis proposes the use of Field Programmable Gate Array (FPGA)-based Programmable System-on-Chips (PSoCs) architectures, as they are ideal computing platforms for flexible data acquisition and processing. However, as the amount of data increases, so does its value, placing the security of these platforms into focus. The contribution of this thesis is, therefore, the design and evaluation of a holistic security architecture that addresses crucial aspects of PSoC security, particularly during key provisioning, boot, and remote updates. In the following, the main contributions of this work are summarized, followed by a perspective of possible future work.

6.1 Summary

Both downsides and upsides must first be answered in terms of key security issues to protect sensitive data from potential attacks on PSoC-based edge platforms. This is especially true since, due to the tight integration of programmable hardware and software components, FPGA-based edge platforms open up attack vectors that are not present in traditional computing systems. For example, an attacker with physical access to non-volatile memory could manipulate the loading of an initial FPGA configuration, negatively affecting the entire system. In addition, attacks targeting cryptographic key storage to compromise the security of secret keys threaten mandatory trust assump-
tion required for unique and genuine device authentication. In fact, undermining the integrity or confidentiality of cryptographic keys poses the most critical risk since the entire system's security is typically based on these keys. As a remedy, in Chapter 3, this work proposes a novel digitally tunable PUF circuit as an alternative to permanent key storage for edge devices operating in an untrusted environment. In particular, silicon PUFs make it possible to convert natural variations in transistor characteristics into a unique, FPGA-immanent secret. Here, addressable shift registers available within the FPGA are used to exploit, for the first time, tunable signal propagation times to generate PUF responses. In doing so, it can be shown that these freely configurable signal propagation times can be adapted for a given PUF circuit to achieve both better uniqueness and reliability properties without losing its fundamental unpredictability and non-copyability properties. To this end, completely novel analyses and evaluations of the effects of temperature variations concerning the stability of these configurations are presented. The results highlight that the proposed approach provides significantly better results in uniqueness, reliability, and robustness compared to existing static PUF designs. Moreover, it can be shown that device-specific PUF configurations can be provided that achieve savings in terms of helper data storage requirements and error correction time required for cryptographic key generation.

The initial hardware configuration of an SRAM-based FPGA combined with the software initialization of the processor after power-on is a unique feature of PSoCs. However, given the unsupervised environment edge devices are operated in, an attacker could tamper with an initial hardware/software configuration loaded from external non-volatile memory to maliciously manipulate the system's integrity. As a remedy, in Chapter 4, this work extends the proposed PUF-based key generation method to an innovative approach for secure data communication with external non-volatile memory devices. Here, the reconfigurable logic of the FPGA serves as a hardware-based trust basis for security-critical authentication and integrity checking operations during data transfers between the PSoC and the external memory device. In this process, a hardware-protected security unit called Trusted Memory-Interface Unit (TMIU) with access to the non-volatile memory is first loaded into the FPGA when the system is started. Then, after appropriate authentication of the memory device, simultaneous decryption and verification of data is performed, making it possible to ensure that the PSoC is booted into a secure operating state. In this way, it becomes possible for the first time to securely load and store both FPGA-based PSoC configurations and other sensitive data on a large scale across reboot cycles. Furthermore, it is shown that the read and write speed of the TMIU is mainly determined by the time required to wait for input/output operations to complete. The evaluated design demonstrates that an optimal data throughput to the storage device is achieved since transfers are possible at line-rate despite the extra overhead imposed by the cryptographic procedures executed in hardware.

Once an initial system configuration has been securely loaded, remote maintenance operations such as updates or upgrades are required to ensure the secure deployment of the often globally distributed edge platforms throughout their lifecycle. Especially here,
immunity requirements such as the authenticity of the update provider and the integrity and confidentiality of the content communicated over an untrusted network are essential. To that end, in Chapter 5, this work proposes a secure communication protocol that allows to first uniquely verify the authorization of an external update provider and, based on this authentication, establish a protected channel for secure data transmission. The proposed update protocol can also guarantee the confidentiality of the cryptographic keys used in the process, even if the PSoC’s software were to be compromised. Furthermore, it can be shown that through partial yet secure reconfiguration of cryptographic FPGA modules only a minor and most importantly temporary (i.e., during the update process) use of logic resources is required.

6.2 Future Directions

The methodologies proposed in this work provide fundamental approaches and mechanisms necessary for a broad and, particularly, secure adoption of PSoC-based compute platforms for edge applications and systems. These contributions can serve as a foundation for several directions of future research, some of which are outlined below.

The concept of the proposed response-tunable PUF design presented in Chapter 3 serves as the backbone for the security schemes offered in this work. Therefore, the PUF design should also be approved for other FPGA types to enable a comprehensive vendor-independent adoption. Considering the PUF’s tuneability, even a dedicated Application-Specific Integrated Circuit (ASIC) implementation can be envisioned as an implementation to be explored. To further improve the robustness of such silicon PUFs against environmental changes, investigating the applicability of a self-configuring calibration approach could be another future direction. Here, calibration steps that independently consider the influence of aging and temperature effects could be researched to enable long-term calibration strategies to preserve the uniqueness and reliability of a given PUF circuit over time. Investigating such dynamic tuning strategies to improve the PUF’s response characteristic in favor of aging effects could serve as the first step in this direction.

In future work, a deeper integration of the dynamic update/upgrade services into the TMIU architecture that guards the secure boot function is conceivable. This integration could result from a combination of the TMIU and TUU design, which would further reduce hardware resources since cryptographic modules for symmetric de- and encryption as well integrity verification could be time-multiplexed. Both hardware designs could even be extended to provide hardware-based protection mechanisms that include external main memory security through encryption and verification. Specifically for the update protocol proposed in Chapter 5, future work should explore strategies for fail-safe fallback modes in the event of failed updates.

In general, further research is required to evaluate the vulnerability to side-channel attacks for all proposed security mechanisms. In particular, it would have to be inves-
tigated whether the number of power traces that an attacker could collect from the cryptographic operations performed inside the TMIU during a boot process or from the TUU during an update process, respectively, would be sufficient to compromise the confidentiality of cryptographic keys. If this were the case, such side-channel vulnerabilities would require additional countermeasures such as obfuscation and key-rolling, introducing corresponding time and resource overheads.

Last but not least, similar additional efforts are required if the PSoC’s threat model is to be extended to provide protection against the compute power of quantum computers. In this case, a re-evaluation of the proposed cryptographic schemes would be necessary. While the impact on the presented symmetric cryptographic methods should be manageable, since they could be secured by doubling the key length, the Elliptic Curve (EC)-based asymmetric cryptography used in the secure update protocol needs to be replaced entirely to resist attacks enabled by quantum computers. Suppose all underlying elliptic curve operations were replaced. In this case, the protocol and the proposed reconfiguration flow could be retained unchanged. Still, the approach would need to be re-evaluated regarding latency, power consumption, and hardware requirements for an alternative Post-Quantum Cryptography (PQC) algorithm.
Sicherheitstechniken für die Datenverarbeitung an der Edge auf HW/SW-programmierbaren SoC-Architekturen
Zusammenfassung

Im Rahmen dieser Arbeit galt es, geeignete Sicherheitstechniken (engl. Security) für den Einsatz sogenannter Hardware- und Software-programmierbarer engl. System-on-Chips (SoCs) für die dezentrale Datenverarbeitung am Rande des Netzwerks (engl. Edge) zu untersuchen.


engl. Physical Unclonable Functions (PUFs) als vielversprechende Alternative zur permanenten Schlüsselspeicherung erwiesen. PUFs sind physikalische Einheiten, die natürlichen Variationen in der Transistorbeschaffenheit in eine definierte Ausgabe umwandeln, aus der kryptografische Schlüssel abgeleitet werden können.


Nach dem sicheren Laden einer initialen Systemkonfiguration sind wiederum Fern-
wartungsvorgänge (engl. Remote Updates) erforderlich, um den sicheren Einsatz der
oft global verteilten Edge-Plattformen über ihren gesamten Lebenszyklus hinweg zu
gewährleisten. Gerade hier sind Immunitätsanforderungen, wie die Authentizität des
Update-Anbieters sowie die Integrität und Vertraulichkeit der über ein nicht vertrau-
enswürdiges Netzwerk kommunizierten Inhalte, essenziell. Der dritte Beitrag dieser
Arbeit beschreibt daher ein Kommunikationsprotokoll für das sichere Senden solcher
Remote Updates. Nach der eindeutigen Verifizierung eines externen Update-Anbieters
wird dabei ein geschützter Kanal für die sichere Datenübertragung eingerichtet. Das
Alleinstellungsmerkmal dieses vorgeschlagenen Aktualisierungsprotokolls ist, dass die
dabei zum Einsatz kommenden kryptografischen Schlüssel nur während des Update-
Verfahrens und nur innerhalb der Hardware verfügbar sind. Das Besondere, dieser
Schutz gilt selbst dann, wenn die Software des SoCs kompromittiert werden sollte. Hin-
sichtlich des Ressourcenbedarfs gilt dabei, dass mittels partieller, aber dennoch sicherer
Rekonfiguration von kryptografischen FPGA-Modulen eine geringe und nur temporäre
(d. h. während des Aktualisierungsprozesses) Inanspruchnahme der Logikressourcen
notwendig ist.


Bibliography


Author’s Own Publications


[SLW*18] Franz-Josef Streit, Martin Letras, Stefan Wildermann, Benjamin Hackenberg, Joachim Falk, Andreas Becher, and Jürgen Teich. Model-Based Design Automation of Hardware/Software Co-Designs for Xilinx Zynq PSoCs. In 13th International Conference on Reconfigurable Computing and FPGAs (ReConFig), pages 1–8. IEEE, 2018. DOI: 10.1109/RECONFIG.2018.8641736.

[PSB*19] Patrick Plagwitz, Franz-Josef Streit, Andreas Becher, Stefan Wildermann, and Jürgen Teich. Compiler-Based High-Level Synthesis of Application-Specific Processors on FPGAs. In 14th International Conference on Reconfigurable Computing and FPGAs (ReConFig), pages 1–8. IEEE, 2019. DOI: 10.1109/ReConFig48160.2019.8994778.


# Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
<td>22</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application-Specific Integrated Circuit</td>
<td>2</td>
</tr>
<tr>
<td>ASR</td>
<td>Addressable Shift Register</td>
<td>13</td>
</tr>
<tr>
<td>BER</td>
<td>Bit Error Rate</td>
<td>39</td>
</tr>
<tr>
<td>BRAM</td>
<td>Block RAM</td>
<td>14</td>
</tr>
<tr>
<td>CAN</td>
<td>Controller Area Network</td>
<td>15</td>
</tr>
<tr>
<td>CBC</td>
<td>Cipher Block Chaining</td>
<td>23</td>
</tr>
<tr>
<td>CLB</td>
<td>Configurable Logic Block</td>
<td>13</td>
</tr>
<tr>
<td>CMOS</td>
<td>Complementary Metal-Oxide Semiconductor</td>
<td>40</td>
</tr>
<tr>
<td>CPS</td>
<td>Cyber-Physical System</td>
<td>1</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
<td>90</td>
</tr>
<tr>
<td>CS</td>
<td>Cloud Server</td>
<td>100</td>
</tr>
<tr>
<td>DPA</td>
<td>Differential Power Analysis</td>
<td>19</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processor</td>
<td>2</td>
</tr>
<tr>
<td>EC</td>
<td>Elliptic Curve</td>
<td>103</td>
</tr>
<tr>
<td>ECB</td>
<td>Electronic Code Book</td>
<td>23</td>
</tr>
<tr>
<td>ECC</td>
<td>Elliptic Curve Cryptography</td>
<td>25</td>
</tr>
<tr>
<td>ECDHE</td>
<td>Elliptic Curve Diffie-Hellman Ephemeral</td>
<td>105</td>
</tr>
<tr>
<td>ECDSA</td>
<td>Elliptic Curve Digital Signature Algorithm</td>
<td>109</td>
</tr>
<tr>
<td>FF</td>
<td>Flip-Flop</td>
<td>13</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
<td></td>
</tr>
<tr>
<td>HKMG</td>
<td>High-K Metal Gate</td>
<td></td>
</tr>
<tr>
<td>HMAC</td>
<td>Keyed-Hash Message Authentication Code</td>
<td></td>
</tr>
<tr>
<td>IC</td>
<td>Integrated Circuit</td>
<td></td>
</tr>
<tr>
<td>ICAP</td>
<td>Internal Configuration Access Port</td>
<td></td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
<td></td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual Property</td>
<td></td>
</tr>
<tr>
<td>JTAG</td>
<td>Joint Test Action Group</td>
<td></td>
</tr>
<tr>
<td>KDF</td>
<td>Key Derivation Function</td>
<td></td>
</tr>
<tr>
<td>LUT</td>
<td>Lookup Table</td>
<td></td>
</tr>
<tr>
<td>MAC</td>
<td>Message Authentication Code</td>
<td></td>
</tr>
<tr>
<td>MBR</td>
<td>Master Boot Record</td>
<td></td>
</tr>
<tr>
<td>MOSFET</td>
<td>Metal-Oxide-Semiconductor Field-Effect Transistor</td>
<td></td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
<td></td>
</tr>
<tr>
<td>NVM</td>
<td>Non-Volatile Memory</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
<td></td>
</tr>
<tr>
<td>PBKDF</td>
<td>Password-Based Key Derivation Function</td>
<td></td>
</tr>
<tr>
<td>PFS</td>
<td>Perfect Forward Secrecy</td>
<td></td>
</tr>
<tr>
<td>PLC</td>
<td>Programmable Logic Controller</td>
<td></td>
</tr>
<tr>
<td>PQC</td>
<td>Post-Quantum Cryptography</td>
<td></td>
</tr>
<tr>
<td>PSoc</td>
<td>Programmable System-on-Chip</td>
<td></td>
</tr>
<tr>
<td>PUF</td>
<td>Physical Unclonable Function</td>
<td></td>
</tr>
<tr>
<td>RO</td>
<td>Ring Oscillator</td>
<td></td>
</tr>
<tr>
<td>RSA</td>
<td>Rivest-Shamir-Adleman</td>
<td></td>
</tr>
<tr>
<td>SDIO</td>
<td>Secure Digital Input/Output</td>
<td></td>
</tr>
<tr>
<td>SHA</td>
<td>Secure Hash Algorithm</td>
<td></td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>SoC</td>
<td>System-on-Chip ........................................ v</td>
<td></td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer ...................................... 18</td>
<td></td>
</tr>
<tr>
<td>TDEA</td>
<td>Triple Data Encryption Algorithm ......................... 22</td>
<td></td>
</tr>
<tr>
<td>TLS</td>
<td>Thermal Laser Stimulation .................................. 33</td>
<td></td>
</tr>
<tr>
<td>TMIU</td>
<td>Trusted Memory-Interface Unit .............................. vi</td>
<td></td>
</tr>
<tr>
<td>TRNG</td>
<td>True Random Number Generator ............................. 108</td>
<td></td>
</tr>
<tr>
<td>TUU</td>
<td>Trusted Update Unit ....................................... 8</td>
<td></td>
</tr>
<tr>
<td>WCET</td>
<td>Worst-Case Execution Time .................................. 74</td>
<td></td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>$\varepsilon (D, k_s)$</td>
<td>Function for sym. encryption of data $D$ with secret key $k_s$</td>
<td>23</td>
</tr>
<tr>
<td>$\sigma (D, k)$</td>
<td>Function for asym. authentication of data $D$ with a public/secret key pair $k \in {k_p, k_s}$</td>
<td>24</td>
</tr>
<tr>
<td>$\mu (D)$</td>
<td>Function for cryptographic hashing of data $D$</td>
<td>29, 143</td>
</tr>
<tr>
<td>$\eta (D, k_s)$</td>
<td>Function for message authentication of data $D$ with secret key $k_s$</td>
<td>29</td>
</tr>
<tr>
<td>$PBKDF_{\eta,CNT} (w, S, Z)$</td>
<td>Function for key derivation of length $Z$ from data $w$ and salt $S$</td>
<td>30</td>
</tr>
<tr>
<td>$D$</td>
<td>Set of raw data elements</td>
<td>143, 23</td>
</tr>
<tr>
<td>$E$</td>
<td>Set of encrypted data elements</td>
<td>143, 23</td>
</tr>
<tr>
<td>$S$</td>
<td>Set of data with high randomness e.g., a salt</td>
<td>30</td>
</tr>
<tr>
<td>$U$</td>
<td>Update data</td>
<td>143, 104</td>
</tr>
<tr>
<td>$ID_{SoC}$</td>
<td>Identifier of SoC available from an eFuse register</td>
<td>31, 143</td>
</tr>
<tr>
<td>$ID_{NVM}$</td>
<td>Identifier of NVM available from an eFuse register</td>
<td>143, 86</td>
</tr>
<tr>
<td>$T_A$</td>
<td>List of ambient temperatures</td>
<td>65</td>
</tr>
<tr>
<td>$H$</td>
<td>List of helper data</td>
<td>70</td>
</tr>
<tr>
<td>$C$</td>
<td>List of cipher data</td>
<td>30</td>
</tr>
<tr>
<td>$M$</td>
<td>List of bitmask data</td>
<td>73</td>
</tr>
<tr>
<td>$V$</td>
<td>List of update versions</td>
<td>143, 104</td>
</tr>
<tr>
<td>$\mathcal{C}$</td>
<td>Tuple of a CHOICE PUF configuration</td>
<td>55</td>
</tr>
<tr>
<td>$\mathcal{D}$</td>
<td>Tuple of ECC domain parameters</td>
<td>28</td>
</tr>
<tr>
<td>$\mathcal{P}$</td>
<td>Physical system with uniqueness properties e.g., a PUF</td>
<td>37</td>
</tr>
<tr>
<td>$F$</td>
<td>Field with finite number of elements</td>
<td>26</td>
</tr>
<tr>
<td>$a$</td>
<td>Address vector of ASR</td>
<td>52</td>
</tr>
<tr>
<td>$P$</td>
<td>Pattern for a CHOICE PUF configuration</td>
<td>52</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>Size of helper data in bytes</td>
<td>72</td>
</tr>
<tr>
<td>HD</td>
<td>Number of bits differing (Hamming distance)</td>
<td>38</td>
</tr>
<tr>
<td>I</td>
<td>Number of sensor interfaces</td>
<td>12</td>
</tr>
<tr>
<td>M</td>
<td>Number of PUF measurements</td>
<td>38</td>
</tr>
<tr>
<td>N</td>
<td>Number of evaluated devices</td>
<td>38</td>
</tr>
<tr>
<td>Z</td>
<td>Number of elements in vector</td>
<td>22</td>
</tr>
<tr>
<td>R</td>
<td>Number of elements in random data</td>
<td>30</td>
</tr>
<tr>
<td>E</td>
<td>Number of helper data entries</td>
<td>72</td>
</tr>
<tr>
<td>CNT</td>
<td>Number of fixed iteration counts</td>
<td>30</td>
</tr>
<tr>
<td>K</td>
<td>Number specifying the finite field $\mathbb{F}_{2^K}$</td>
<td>27</td>
</tr>
<tr>
<td>$T_A$</td>
<td>Value of ambient temperature in °C</td>
<td>45</td>
</tr>
<tr>
<td>$T_J$</td>
<td>Value of junction temperature in °C</td>
<td>46</td>
</tr>
<tr>
<td>$m$</td>
<td>Index of PUF measurement</td>
<td>37</td>
</tr>
<tr>
<td>$m_v$</td>
<td>Index of voting</td>
<td>69</td>
</tr>
<tr>
<td>c</td>
<td>Index of a CHOICE configuration</td>
<td>60</td>
</tr>
<tr>
<td>b</td>
<td>Index of device</td>
<td>37</td>
</tr>
<tr>
<td>v</td>
<td>Index of update version</td>
<td>144, 104</td>
</tr>
<tr>
<td>$r_{b,c}^m \in {0, 1}^Z$</td>
<td>PUF response of measurement $m$ with CHOICE config. $c$ on board $b$</td>
<td>60</td>
</tr>
<tr>
<td>$r_{b,c}^* \in {0, 1}^Z$</td>
<td>PUF nominal response of CHOICE config. $c$ on board $b$</td>
<td>60</td>
</tr>
<tr>
<td>$r_{b,c}^i \in {0, 1}^Z$</td>
<td>PUF noisy nominal response of CHOICE config. $c$ on board $b$</td>
<td>69</td>
</tr>
<tr>
<td>$k \in {0, 1}^Z$</td>
<td>Cryptographic key of length $Z$</td>
<td>30, 144</td>
</tr>
<tr>
<td>$k_s \in {0, 1}^Z$</td>
<td>Secret key of length $Z$</td>
<td>23, 144</td>
</tr>
<tr>
<td>$k_p \in {0, 1}^Z$</td>
<td>Public key of length $Z$</td>
<td>24</td>
</tr>
<tr>
<td>w</td>
<td>Secret not directly applicable as cryptographic key, e.g., a password</td>
<td>30</td>
</tr>
<tr>
<td>$\Delta T$</td>
<td>Temperature range in K</td>
<td>7</td>
</tr>
<tr>
<td>$\tau$</td>
<td>Propagation delay</td>
<td>42</td>
</tr>
<tr>
<td>$\delta$</td>
<td>Width of glitch signal</td>
<td>49</td>
</tr>
<tr>
<td>$\delta_{\text{min}}$</td>
<td>Minimal width of glitch signal</td>
<td>52</td>
</tr>
<tr>
<td>R</td>
<td>Electrical resistance</td>
<td>42</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>$C$</td>
<td>Electrical capacitance</td>
<td>42</td>
</tr>
<tr>
<td>$V$</td>
<td>Electrical voltage</td>
<td>144, 42</td>
</tr>
<tr>
<td>$\parallel$</td>
<td>Concatenation operator</td>
<td>145</td>
</tr>
</tbody>
</table>
A
- aging ........................................ 46
- attacks ....................................... 16
  - hardware .................................. 19
  - invasive attack ............................ 17
  - logical .................................... 17
  - network .................................. 18
  - non-invasive attack ....................... 17
  - physical .................................. 17
  - semi-invasive attack ...................... 17
  - software .................................. 18
- availability .................................. 16

B
- Bit Error Rate (BER) ......................... 39
- bitstream .................................... 14
- boot ......................................... 21, 77
  - image ...................................... 79
  - loader first ................................ 79
  - loader second ................................ 79
- bug
  - Heartbleed .................................. 102
  - Starbleed .................................. 82

C
- cloud computing .............................. 1
- confidentiality ................................ 16
- Configurable Logic Block (CLB) ......... 13

D
- device DNA .................................. 31, 86
- digest ....................................... 28
- discrete logarithm problem .................. 25
- dopants ..................................... 43

E
- edge computing ............................... 10
- elliptic curve ................................ 26
  - point addition ............................. 26
  - point doubling ............................ 26
  - scalar point multiplication ............... 26

F
- Field Programmable Gate Array (FPGA) 12
  - firmware .................................. 102
  - FPGA Vendor ............................... 31
    - Intel/Altera ............................... 32
    - Xilinx .................................... 31
- fuzzy extractor .............................. 70, 87, 109

G
- glitch ......................................... 49

H
- hardware Trojan ............................ 19, 83
- hash ......................................... 28

I
- integrity ..................................... 16
  - inter-die .................................. 38
  - intrinsic .................................. 44

K
- Keccak ....................................... 29
- key .......................................... 30
  - derivation .................................. 28
  - generator .................................. 30, 73
  - key storage .................................. 31
  - battery-backed RAM ......................... 31
  - eFuses ..................................... 31
Index

L
laser fault injection 33
lithography 43
Lookup Table 13

M
memory II
main memory (DDR) II
non-volatile memory (NVM) II

N
nominal response 38

O
operating system II
Linux 79
optical 17

P
Physical Unclonable Function 36
prime factorization 25
process variations 43
Programmable System-on-Chip (PSoC) II

Q
quantum II, 113, 120
computer II, 113
secure II, 114

R
random
function 37
number 108
physical phenomena 35
session key 108
timing characteristics 37
reliability 63
Ring Oscillator 47, 107
robustness 64
roughness
line-edge 44
line-width 44

S
salt 30, 72, 73
sector 79
side-channel 18, 89, 119
signature 24

temperature 45
ambient 45
instability 47
junction 46
variations 45, 66
threshold voltage 41
transistor 41
dimensions 43
derain 42
gate 41
source 41

U
uniformity 60
uniqueness 38
update 100

V
voting 69

W
warm start address (WB-STAR) 82
Worst-Case Execution Time (WCET) 74

X
Xilinx
Artix 41, 44, 46
Virtex 79