Work Package 2:

Requirements Analysis

for an HSM, EDA Tools and a Demonstrator Setup

Version	1.0
Project Coordination	IHP GmbH
Date	December 3, 2021

SPONSORED BY THE

Authors

- Elektrobit Automotive GmbH:
  - Martin Böhner
- Fraunhofer SIT:
  - Norman Lahr
  - Julian Wälde
- IAV GmbH:
  - René Rathfelder
- Hochschule RheinMain:
  - Steffen Reith
  - Tim Henkes
- TU Berlin:
  - Tuba Kiyan
  - Arnd Weber
- Ruhr-Universität Bochum:
  - Pascal Sasdrich
  - Fabian Buschkowski
- IHP:
  - Norbert Herfurth
  - Markus Ulbricht
- DFKI:
  - Christoph Lüth
  - Milan Funck
  - Vladimir Herdt

Please quote as:
Böhner, Martin; Buschkowski, Fabian; Funck, Milan; Henkes, Tim; Herdt, Vladimir; Herfurth, Norbert; Kiyan, Tuba; Lahr, Norman; Lüth, Christoph; Rathfelder, René; Reith, Steffen; Sasdrich, Pascal; Ulbricht, Markus; Wälde, Julian; Weber, Arnd: Requirements Analysis for an HSM, EDA Tools and a Demonstrator Setup. Project HEP, 2021, `https://hep-alliance.org/`

Project Coordination
Dr. Norbert Herfurth
IHP GmbH – Innovations for High Performance Microelectronics
Leibniz-Institut für innovative Mikroelektronik
Im Technologiepark 25
15236 Frankfurt (Oder)
Germany

Phone +49 335 5625525
Mail herfurth@ihp-microelectronics.com

Förderkennzeichen ME1ZEUS012

Associated Partners
Bosch
CARIAD (a Volkswagen Group Company)
HENSOLDT Cyber
Hyperstone
Swissbit Germany

Contents

1 Terminology

This section provides an overview of abbreviations and terms used in this document.

Abbreviation	Name	Explanation
ASIC	Application Specific Integrated Circuit	An integrated circuited specified for a given application.
CA	Certification Authority	An entity in a PKI that issues digital certificates.
CAN	Controller Area Network	CAN is a serial bus system used in the automotive and automation domain. The bus data is transmitted with a differential signal for robustness reasons.
DFT	Design for Testing	Integrated circuit design techniques that add testability features to a hardware product design. .
DRC	Design Rule Check	A geometric constraint imposed on circuit board, semiconductor device, and integrated circuit designers to ensure their designs function properly, reliably, and can be produced with acceptable yield.
ECU	Electronic Control Unit	ECU is the general term for an electronic component in the in-vehicle network with certain functions required by the overall architecture of the vehicle for the correct usage of the vehicle.
FPGA	Field Programmable Grid Array	Integrated circuit that can be configured by a customer via Hardware Description Languages.
GDSII	Graphical Data Stream Information Interchange	is a database file format which is the industry standard for data exchange of integrated circuit or IC layout artwork.
HDL	Hardware Description Language	Specialized Computer language to describe the structure and behavior of electronic circuits.
HCL	Hardware Construction Language	An extended Computer language to build software that creates hardware structures during execution using an HDL as output.

HSM	Hardware Security Module	Additional component offering security services to the device or system: In the automotive domain an HSM is an additional IP core inside a chip or microcontroller that offers a security enclave with cryptographic services to the main processor of the chip or microcontroller.
ISA	Instruction Set Architecture	An abstract model of a computer also called the computer architecture.
IP	Intellectual property	A reusable unit of logic, cell, or integrated circuit layout design that is the intellectual property of one party.
LVS	Layout versus Schematic	The class of electronic design automation (EDA) verification software that determines whether a particular integrated circuit layout corresponds to the original schematic or circuit diagram of the design.
RISC	Reduced Instruction Set Computer	A computer with a small, optimized instruction set.
RISC-V	Reduced Instruction Set Computer V	An open standard Instruction Set Architecture based on RISC principles.
RTL	Register-transfer-level	A design abstraction which is used in hardware description languages to create high-level representations of a circuit.
SCA	Side Channel Attack	Any attack based on information gained from the implementation of a computer system.
TCG	Trusted Computing Group	An industrial standardization body for establishing open standards for the trusting computing platform.
TPM	Trusted Platform Module	An additional security chip with security features specified by the TCG to provide security features to the host system.

Table 1.1: Overview of the terminology used in this report.

2 Introduction

2.1 Introductory note

This is a draft report by project HEP (Hardening the Value Chain through Open Source, Trustworthy EDA Tools and Processors). It contains chapters about our motivation for designing an open, secure HSM, about our use case "automotive sensor communication", about the selection of the VexRiscv-design and about our choice of open hardware development tools. The final chapter contains our verification plan. The authors thank the Associated Partners for feedback.

2.2 Motivation for Research on Open-Source Hardware

The open-source approach to digital hardware design is relatively new and contrasts sharply with the way things are done today. Currently, it is good industrial practice to make all information about the design process of hardware available only under NDA and to work with patents. This has some well-known disadvantages. Innovations do not spread as quickly as possible, because common knowledge and new ideas are not shared. This restriction prevents the development of a culture of collaboration between developer and user. Ultimately, the knowledge available to the user cannot be used by the developer because the improvements from the user community do not flow back. There are further disadvantages for the user. Due to the commitment to a manufacturer, special extensions cannot simply be made by the user. This is a problem especially for products with a small number of units, as these are usually economically uninteresting for the original manufacturer. Similar disadvantages occur if the products have a very long-life cycle and the original manufacturer has long since discontinued the products.

The problems are comparable to the recent history of the software industry, which suffered for a long time from similar problems. In the 1980s, the MIT AI lab switched to a PDP-10 computer infrastructure that used a propriety operating system from DEC. According to Richard Stallman, this made small repairs, simple fixes and improvements impossible and left the AI Lab completely dependent on DEC's service team. It quickly became clear that not only a free operating system would be beneficial, but also appropriate development tools (e.g. compilers, editors) must be available to drive new developments. A change here was brought by the GNU project (and similar efforts), which made a multitude of new business models possible because suddenly the knowledge about UNIX, databases, networking and other fundamental technologies became accessible to a broad mass of initially young developers, at low costs. But not only an operating system like Linux or the flavors of BSD-Unix have played an important role here. Especially the cheap and comfortable development tools were of

extreme importance. For example, we know today that many services of the Internet would not even exist without the C compiler gcc and other tools established by the GNU project. Good examples are the free Apache web server or server operating systems like Debian, which form the backbone of the Internet.

Through open-source software, entire generations of developers were equipped with deep skills around the Internet, who had initially pursued the topic only for hobby reasons (cf. Linus Torvalds and the development of Linux). Only later it turned out that this development model has large advantages with many applications, since knowledge and development costs can be shared, in order to be able to tackle large projects together. At the same time, the universities were able to take up this development in research and teaching, since the necessary information was, and still is, freely accessible and thus strengthen the effect by training new young developers.

This results in major advantages, which take place especially for basic technologies (cf. [TSN17]), where competitors are not in direct competition, but which are essential for a successful product:

- Development costs are shared between competitors. This applies in particular to basic technologies that are essential for product development but which cannot be seen by an end user (e.g., operating system kernels, communication protocols).
- Agreements on de-facto standards are established more easily, since there is no interest in leveraging them through secret and proprietary developments.
- Over longer periods of time a larger developer community becomes available, which makes a multiplicity of projects possible and helps to reduce personnel costs.
- Trade wars and trade restrictions can be avoided, since the development documents are freely accessible and thus not subject to arbitrary regulations.

These advantages have revolutionized the software industry market and brought us new business models, mass-market applications such as cloud computing, social media and numerous Internet-based services. Certainly, all of these technologies could also have been developed with a closed-source approach. However, certainly at a higher price and at the expense of a lower development speed.

Almost 40 years after the founding of the GNU project and the start of the Free Software Foundation, it is becoming apparent that a similar approach can be taken in the field of building hardware. The first steps were free development tools for dealing with (commercial) FPGAs and their extension for the design of ASICs. Here, the role of gcc and the GNU coreutils may be taken over by tools like Yosys (synthesis) and NextPNR (Place&Route). Almost simultaneously, the open ISA RISC-V was developed. This led to a large number of different RISC-V implementations (of varying quality, size and efficiency) becoming available under open-source licenses. These efforts have the clear potential to enable the development of large hardware projects soon, much like the GNU tools have helped Linux or BSD-Unix to achieve success.

Major market players have already recognized this new development. For example, Western Digital is integrating their RISC-V implementation SweRV as a new technology

into their products and even giving their developments back to the community. Another example is NVIDIA. They plan to replace their Falcon microprocessor, used as an embedded control instance for their GPUs, by a RISC-V processor. Interestingly, one of their goals is to ensure data integrity of sensors for automotive applications (cf. [SX20]) and they use Spark/ADA for formal verification. Furthermore, there are reports that Intel wants to take over the RISC-V pioneer SiFive. One of the reasons seems to be that Intel wants to offer solutions for the automotive market - SiFive licenses IP for high-end RISC-V applications to Renesas (cf. [Dah21]).

OpenTitan plans to build an open-source silicon root of trust. Hence, the goals of OpenTitan are similar to those of HEP. However, HEP will go a step further by using open-source tools for synthesizing RTL descriptions, whenever possible. This makes it possible to intervene in the process of generating a circuit in order to be able to implement hardening measures automatically. According to the current state of knowledge, such an approach would be much more difficult to implement for OpenTitan, since the project is based on SystemVerilog and no open development tools exist for this at the current time.

All these current developments show that both the basic ideas of HEP and the application examples have been carefully selected. In contrast, others perceive RISC-V as a threat to their business model. For example, ARM warned users about RISC-V technology in much the same way as SCO-Unix did in the case of Linux (cf. [Hru18]).

In addition to the well-known advantages from the software world, the open development approach of hardware will have other benefits as well. Long-term availability of hardware, especially low-volume products, would also be strengthened, which is a major benefit for some industries like home automation, critical infrastructure and healthcare with extremely long-lived products. If such a manufacturer discontinues a product, the required goods can be produced by others. In addition, it is even possible to take the development of the otherwise supplied parts into one's own hands, which strengthens the flexibility of the production process.

Due to the multitude of advantages of open-source hardware, project HEP would like to contribute to this emerging trend. This has major advantages in the context of IT security. For example, certification processes become easier, even mathematically provably correct code can be produced and its development costs are shared. Hardening measures against attacks such as side-channel attacks or hardware Trojans can be more easily developed and errors can be more easily ironed out. The design can be more easily tested, improved, and further developed. Such measures and techniques can be implemented in industry or in academic environments or both together. Finally, user trust increases because they can check open-source hardware designs and, at some cost, even products.

HEP therefore aims to design a key component of IT-security, namely an open-source hardware security module (HSM, similar to Trusted Platform Modules, Secure Enclaves etc.). In doing so, all steps of the hardware design will be done exclusively with open-source tools, if possible, and the resulting design will be made open. Some tools will remain proprietary, for the time being, e.g. the development hardware (PCs) and parts of the hardware production software, e.g. mask generation software. Still, the attack surface will be reduced. To improve the quality of a critical module such as an

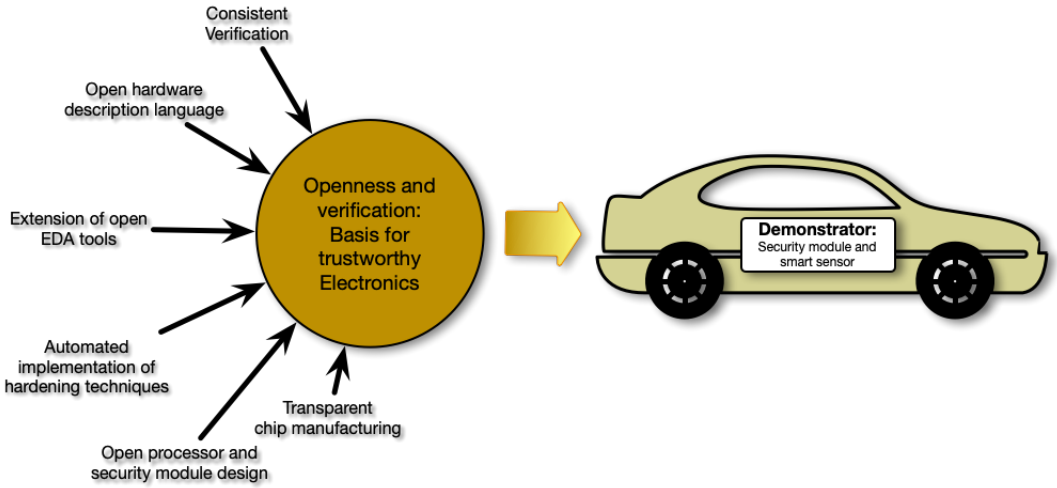

Figure 2.1: Research Topics in Project HEP

HSM, open-source formal verification techniques will also be developed, extended, and integrated into an open hardware description language and the related synthesis-tool. Furthermore, it is planned to check whether formally proven components are still intact when implemented in silicon.

The automotive industry, one of Germany's key industries, was selected as the first application domain to demonstrate the importance and also the feasibility of open-source hardware in industry. In the planned demonstrator, the communication of an automotive sensor will be protected (cf. Figure 2.1)

The consortium is firmly convinced that the strengthening of open-source hardware will help to open up new markets for German companies, from small start-ups to big players. This will lead to a new generation of hardware developers who will be able to implement novel ideas. Hence, this project will thus contribute to strengthening Germany's and everybody else's digital sovereignty. Early birds will have more secure and better certifiable products, less risks of damage to reputation due to vulnerabilities, and all this even at lower costs.

2.3 Project goal

The project VE-HEP will show the feasibility of using open-source tools in a wide range of the digital chip design value chain. This includes the use of an open-source processor design, the use of open-source design tools e.g. for the synthesis or the "place and route", as well as the development of public available automatic implementation schemes to harden the chip design against hardware attacks. The use case to implement all this ambitious goals is the design and fabrication of an open-source hardware security module (HSM) that will be integrated into an automotive application. To improve the quality and trustworthiness of such a critical module, verification schemes for the different design steps will be developed, extended and integrated into the design flow.

2.4 Threats

The increasing complexity of electronic systems for information, communication and entertainment in cars and other industrial products are leading to a growing attack and vulnerability surface. This is true for the designed and manufactured product in use but also for the whole chain of development. In order to secure these implementations there has been a development of regulations and standards, in particular the ISO/SAE 21434 "'Road vehicles - Cybersecurity standards"' which is a collaborative work by ISO and SAE working groups [ISO21]. The fulfillment of this international standard will be required for car manufacturers for type approval in the future. But even with these guidelines and mandatory technical standards there are challenges in the future.

To secure a minimal response time between detecting an exploit and fixing it, manufacturers need to work together and establish solutions for tackling the risks that shared technologies, components or infrastructures can pose. Other problems in the near future are the increasing calculation power in contrast to the long lifetime of industrial products (e.g. cars) or plants, especially in regards of quantum computers.

Furthermore the demand for online services and over the air updates is growing or becoming even mandatory. The prerequisite for such features is to find, fix and prevent security issues, software bugs or adding new features to mitigate vulnerabilities. Nevertheless the increasing connectivity also poses a threat vector for criminal intent. [SVZ19] The growing feature count for infotainment and Car2X solutions with more and more interfaces (often wireless) could lead to a combination of attacks that can penetrate vital functionality of vehicles or industrial applications. The possible incentives for attackers range from stealing property, manipulation or cybercrime incidents like blackmailing. Another problem for manufacturers are warranty claims of products which are reset to manufacturer settings after tempering. Therefore implementations for tampering detection should be developed if a protection is too expensive or not possible. This is also a viable solution for attacks that aim to destroy assets slowly and undetected over long timeframes. [Lan13]

Another problem of manufacturers is that successful reverse engineering and hacks from professionals are often not published. They are kept as a trade secret for offering services like Tuning or being traded in criminal marketplaces. These attacks, often performed with sophisticated side channel attacks [OFl20], zero day exploits, time consuming memory readouts or other attack vectors and the combination of these, are hard to prevent and often neglected or even accepted in current threat and risk assessments [Ustr21]. Possible countermeasures are either technical or socially incentive programs. On the technical side the implementation of fault-injection detection mechanism is a possibility to counter those attacks. On the other hand manufacturers can offer bug rewards and hacking competitions as well as using open source methods which can be tested and improved by a large community.

Supply chain risks are of a different type. For example, natural catastrophes or political turmoils can happen, suppliers may show oligopolistic behavior, third parties might attack the assembly of components or their transport. In such cases, open hardware designs would allow to change suppliers more easily.

3 Use Case Description and Specification

In this chapter we will give a detailed description of the use cases and possible threats.

3.1 Demonstrator Use Cases

This section gives an overview of the use cases regarding the demonstrator for showcasing the practical usage of the designed hardware security module. A communication instance consists of an application microcontroller, a connected hardware security module and can be extended with connected peripheral hardware like sensors or actors dependent on the specific use case. In addition to the hardware, the demonstrator also consists of the firm- and software developed by the consortium. The purpose of the demonstrator is to showcase the functionality of the developed security module and its features. To accompany the use cases, the requirements for the application board, interfaces and peripherals have to be specified. Beside the functionality, the use cases will include test cases and a comparison to similar implementations. The security threats each use case is facing are described as well as how the demonstrator is preventing or mitigating the risks.

The demonstrator setup is divided into multiple instances. One of the two main instances contains a connected sensor for data acquisition, and the other instance is for receiving the communicated messages and other demonstration purposes. Other instances will be used to threaten the functionality and integrity of the sender and receiver instances.

Each instance has an application microcontroller and the connected hardware security module. The sensor will be interchangeable and can even be simulated for testing purposes. The sender and receiver entities are connected via industry standard protocols. An overview of the demonstrator is depicted in Figure 3.1 with corresponding software modules that have to be developed.

3.1.1 Authenticated Communication Between Instances

This use case regards the communication between the two instances of the demonstrator over industry standard protocols like CAN or Ethernet. To ensure that Sender and Receiver are not compromised by a third party, an authentication process has to be used.

To ensure that all communication is authenticated, unauthenticated messages should only be used for establishing the authentication process and should be dropped if they have another purpose. This also eliminates the risk of sending confidential data to malicious entities.

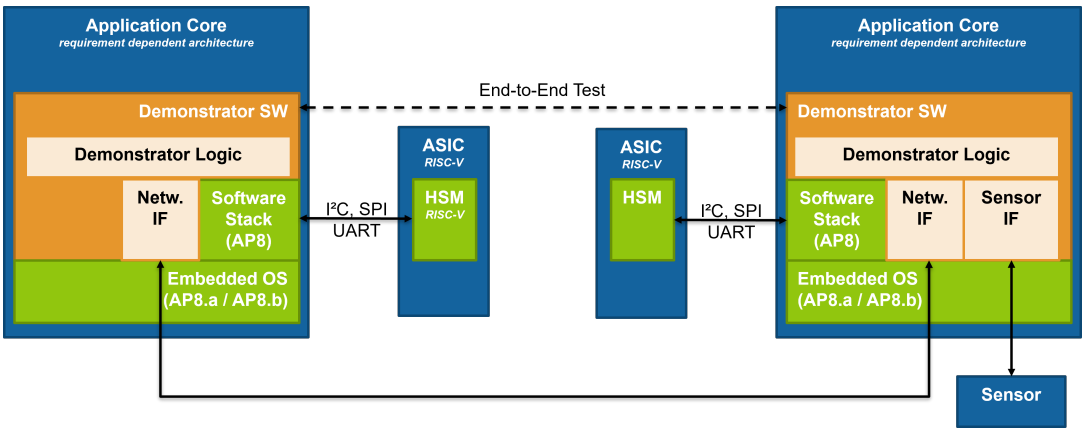

Figure 3.1: Demonstrator Overview with connected HSMs and Sensor

The authenticated communication will be further described in derived use cases of the hardware security module: (3.2.2) Secure Update, as well as (3.2.3) Confidential Data Access.

Application Microcontroller Requirements

- protocol support for CAN interface
- protocol support for SPI interface for HSM connection
- protocol support for JTAG interface for debugging
- desirable protocol support for Ethernet interface
- desirable protocol support for I²C or UART for further interface testing or sensor application

3.1.2 Secure Communication Between Instances

This use case describes the secured communication between instances. After authentication, the messages are encrypted by the sender and will be decrypted by the receiver via the hardware security module. Therefore, systems for symmetric or asymmetric encryption have to be implemented and developed.

The secure communication will be further described in derived use cases of the hardware security module: (3.2.3) Confidential Data Access.

Application Microcontroller Requirements

- sufficient features for debugging and timers
- preferably built in hardware cryptographic functionality (for comparison with the developed security module performance)

3.1.3 Secure Operation

This use case describes test cases for the general operation of the demonstrator and its use of the developed hardware security module for securing it against threats. The functionality of sending and receiving messages, commands and requests as well as changing the implemented modes of operation will be thoroughly tested. This also includes a logging concept and the monitoring of security-relevant operations and events.

Additionally, a concept for manipulation detection will be developed and tested. The manipulation will be realized with different attack scenarios. An attack instance will be connected to the communication bus of the two instances and will try to disturb the operation of both instances, get access to sensitive data, update the devices with malicious software or lock them in an unproductive state.

The interface between the application board and the hardware security module is not part of the security concept. In a final product the security module should be embedded in one integrated circuit and connected to memory and other peripherals via buses inside the circuit.

The secure operation will utilize all use cases described in the Chapter: Hardware Security Module Use Cases.

Application Microcontroller Requirements

- adequate memory for implementing operating system and demonstrator software
- sufficient calculation power for operation, logging and debugging purposes
- industry common debugging interfaces and standards

3.1.4 Data Acquisition

To demonstrate the viability of the hardware security module we acquire data with a sensor which is connected to one application board. Different application sensors will be chosen to demonstrate a broad range of applications to prove the functionality and show the impact of different encryption possibilities. Another showcase will be to demonstrate the functionality of a virtual sensor implementation to easily manipulate data rates and payload lengths.

The developed tests will measure the timing variances between unencrypted and encrypted communication, the different encryption standards and their configurations. Furthermore, we try to compare the developed hardware implementation with proprietary solutions. Threats which are covered by the use case are the possibility of manipulating the data between sender and receiver as well as replay attacks. Additionally, the vulnerability of the instances against overflowing the communication bus and Denial of Service attacks will be featured.

The data aquisition will be further described in the derived use case of the hardware security module: (3.2.6) Message Authentication.

Application Microcontroller Requirements

- ADC for possible analog sensor connection
- sufficient calculation power and memory for virtual sensor implementation
- preferably protocol support for I²C, SPI or UART for possible sensor connection

3.2 Hardware Security Module Use Cases

A Hardware Security Module (HSM) serves as a root of trust for electronic components and their applications in various contexts. The most common and important use cases are explained in the following sections. These and further use cases are already discussed in scope of projects like [QuantumRISC].

3.2.1 Secure Boot

Secure boot is a mechanism to ensure the software integrity of a device. Before code instructions are executed, the secure boot mechanism cryptographically verifies the integrity and authenticity of the code block in memory. The goal is to gain trust in the integrity of the executed software, detect its manipulation during the boot phase of the device, and prevent its execution. Modern devices can contain huge software stacks and are often booted in multiple stages for performance reasons. The secure boot workflow has to verify each boot stage before passing control to it.

The trust anchor is a tamper-protected immutable portion of code which is the first code executed after a reset and has to be trusted. The trust anchor verifies the first boot stage and after succeeding, passes control to it. Forming a chain of trust, each boot stage trusts its predecessor, and verifies the integrity of its successor until the device is booted completely.

There are three secure boot verification schemes: hash-based, MAC-based and signature-based verification. On devices for which booting is a time-critical process, cryptographic Message Authentication Codes (MACs) are often the basis for the secure boot process. In this verification scheme the binary blob containing all executable code instructions and configuration data is stored on the device along with a cryptographic MAC over the blob. The MAC is securely generated with a symmetric key which is only known to the device. During the boot phase of the device, the secure boot workflow verifies the integrity and authenticity of code and configuration by computing the MAC over the binary blob in memory and comparing it to the corresponding MAC stored within secure storage. If the verification fails, the secure boot process enforces active countermeasures like not booting the application core or limiting the availability of cryptographic keys.

HSM Requirements

- Hash computation, MAC computation and/or asymmetric signature verification

- Tamper-protected (secure) storage[1]

3.2.2 Secure Update

The secure update mechanism enforces that only authenticated software is flashed to an application core. Asymmetric signatures prevent that an attacker can flash manipulated software. The signature is deployed along with the software to be flashed. Therefore, it is signed with a private key in a trusted environment. The corresponding public key used for the verification must be enrolled beforehand and stored tamper-protected in secure storage. To flash software, the device usually boots into a dedicated bootloader. There, it receives the software from a defined source (e.g., the communication unit or the onboard diagnosis interface) and downloads the software to the memory of the application core and adjusts the boot flags. The secure software download mechanism is part of the bootloader and performs a signature verification before flashing the software. Only if the signature verification succeeds, the bootloader flashes the software.

Updating software is not always done by the developer of that software. Instead, it may be distributed to intermediaries who then are tasked to update relevant devices. This kind of update helps in cases where a central server would have a too high load, or when the expected environment of the update does not have the required connectivity (be it connection at all, or download speed). Such an update usually needs to be encrypted at rest, i.e. until it is installed on a target device, and therefore it differs from a ”simple” update. The update also needs to be verifiably unmodified and authenticated. Hence, the client can verify that the update is indeed valid.

HSM Requirements

- Asymmetric signature verification
- Tamper-protected storage
- Symmetric or asymmetric decryption
- Tamper-protected secure storage

3.2.3 Confidential Data Access

Servicing a device containing private information should be done in a way such that the private data can only be accessed by authorized clients. This confidential data access mechanism goes two ways, an attacker should not be able to see which data is accessed nor the data itself once it is relayed back. The data is usually accessed by way of a remote server, or a local diagnosis tool. The onboard device would need to be able to verify that whoever is accessing it is authorized to do so. Such an authorization could stem from a (temporary) certificate signed by an authorized authority. The onboard device would then check whether the certificate is still valid and initiate a transfer using it as proof.

[1] Future products based on the project results may need tamper protection of the entire HSM.

HSM Requirements

- Asymmetric signature aka certificate verification
- Symmetric or asymmetric encryption
- (Realtime tracking for temporary certificates)

Note: Even though realtime tracking is required to verify temporary certificates, a tamper-protected RTC will not be implemented as part of this project, as the goal of this project is not feature completeness.

3.2.4 Secure Access Control

The receiver represents an automotive device that restricts access to certain services and data such as flashing. Only authorized entities shall be able to unlock these operations to access the data/services. Typical entities are manufacturers in field returns or car service stations to enroll (updated) firmware. There are multiple ways to implement such an authorization check via a challenge response scheme:

- signature-based verification scheme
- MAC-based verification scheme
- public key encryption scheme
- symmetric encryption scheme

The signature-based scheme with a private and public key pair works as follows: A public-private key pair is generated in a secure environment (e.g., the backend) and the public key is deployed to the receivers (e.g., during production). If an accessing sender entity wants to unlock the receiver, it sends an "Unlock Request" to the receiver. The receiver generates a random seed with enough entropy and a secure bit-length and then sends it back as "challenge" to the sender. The seed shall ensure the freshness to protect the unlock against replay attacks. The sender can prove the possession of the valid private key by signing the challenge seed with the private key that corresponds to the public key stored in the receiver. The sender sends the generated signature back as a "response" to the receiver where the signature gets verified. The receiver verifies the signature with the public key against the original seed. If these match, the receiver can grant the sender access to the services/data.

The first alternative is to exchange the signature scheme with a MAC verification scheme in terms of a MAC generation on sender side and MAC verification on receiver side instead of the respective signature operations. The overall flow of the protocol is in general identical to the signature-based scheme. In case an encryption operation is used for the authorization verification in the challenge response protocol, the overall flow is identical with exception of the specific cryptographic operation changes. Hence, the challenge generated by the receiver is an encrypted random number. The correct response from the sender to this challenge is the correctly decrypted random number.

This random number is then sent back in plain text to the receiver for verification. Additionally, in this setting, the public key operations are performed on the automotive device with a public key, while the private key operations are performed on a trusted tool. In case a symmetric encryption based scheme is used in the challenge response scheme, the secret keys needs to be pre-shared between sender and receiver.

HSM Requirements

- Cryptographically secure RNG
- Tamper-protected (secure) storage
- Asymmetric signature verification OR MAC computation OR asymmetric encryption OR symmetric encryption

3.2.5 Public Key Infrastructure

Various use cases require asymmetric key pairs/certificates.

Asymmetric keys are used to verify the integrity and authenticity of data. It is essential that the private key is only known to the instance owning the private key. It shall not be known to other entities. The corresponding public key of the private and public key pair is distributed to the communication partners, which shall be able to verify the senders authenticity. The device which is sending data uses its private key to generate a signature for the data. It transmits the data along with its signature. The receiver device uses the corresponding public key to verify the signature of the data. Since the signature is generated with the private key that is only known to the sender device, the receiver can verify the authenticity of the sender with this procedure.

Key Pair Generation

Some devices require one or maybe multiple device unique keys. The private key of the device's unique private and public key pair is stored securely in the HSM's persistent memory and shall not leave the device due to security policies. Therefore, the private and public key pair is generated by the device's HSM or stored into its tamper-protected storage during production. The private key can be kept secretly inside the HSM. The public key of this pair is provided to the device's communication partners. It can be used to verify the device's authenticity.

Certificate Verification

Some use cases require the private and public key pair to be generated in the Public Key Infrastructure. In this case, the key pair is certified by a certification authority (CA) in a backend. Each certificate contains an asymmetric key and is signed using the CA's private key. The certificates issued by the CA and the CA's public key certificate are deployed to the embedded device where they are stored securely in the HSM.

Before using a certificate, entities must verify whether it is valid or not. To do so, the device verifies the content of the certificate for validity and whether it is part of

the certificate chain. Therefore, it verifies the signature of the certificate using the public key of the CA certificate. The validity of the CA certificate must be verified in the same manner against the next higher instance of CA certificate or against a root certificate.

HSM Requirements

- Cryptographically secure RNG
- Tamper-protected secure storage
- Asymmetric signature generation
- Asymmetric signature verification
- Asymmetric key pair generation
- (Realtime tracking for temporary certificates)

3.2.6 Message Authentication

Sensor data is what a system perceives of its surroundings or its state. Thus, it is crucial to secure the data in any scenario where an attacker can alter it maliciously. For this, a scheme to generate an authentication token — either a signature or a MAC over the payload — is applied. The sensor generates the token over its data before transmitting it to the application core. Also, data freshness must be ensured in some cases. A sequence number is added to the sensor's payload data. After verifying the signature, the application core checks whether the sequence number is reasonably higher than the last received one to prevent replay attacks.

HSM Requirements

- Tamper-protected (secure) storage
- Asymmetric signature verification OR MAC computation

3.3 Physical Attacks and Hardware-based Threats

3.3.1 Side-Channel Analysis

Since the seminal description of Side-Channel Analysis (SCA) by Paul Kocher [Koc96; KJJ99], this topic has emerged as a serious threat to modern cryptographic implementations. Now, after more than two decades of maturation, intensive academic and industrial research, and continuous progress, secure implementation of cryptographically strong algorithms is still a challenging and open problem.

Fundamental Concepts. In its conceptional appearance, Side-Channel Analysis is a non-invasive concept which uses any information leakage of a physical device to extract or infer sensitive and secret internal information. Due to the fact that any physical device can be observed and measured during the execution of cryptographic algorithms and implementations, adversaries can extract information by any physical means that are applicable. In particular, extensive research over the years has shown that physical sources and effects, such as timing behavior [Koc96], instantaneous power consumption [KJJ99], electromagnetic (EM) radiations and emanations [GMO01], noise emission [GST17], or temperature and heat dissipation [HS13] can serve as unintentional side channel and rich source of information leakage.

Information Extraction. In practice, most Side-Channel Attacks rely on a hypothetical model to estimate the behavior and physical characteristics of a cryptographic device or implementation, e.g., for the timing behavior, instantaneous power consumption, or electromagnetic emanations. Then, using statistical tools and methods, the hypothetical behavior is compared to the actual and observed behavior in order to identify the correct assumptions and infer the corresponding secret or sensitive information. However, the efficiency of this approach is strongly depending on the soundness and precision of the underlying theoretical model and the corresponding assumptions. In particular, obtaining and deriving accurate models for real-world targets and applications is a challenging and non-trivial task. In addition, even if the model and assumptions are sound and accurate, designers and engineers can incorporate additional mechanisms and measures to inhibit unintentional leakage and the extraction of sensitive internals.

Attack Classification. In modern literature, side-channel analysis and attacks are often classified according to the following taxonomy. However, please note that this list is provided without any claim to completeness but focuses on most common and major categories of side-channel attacks. For this, side-channel attacks can be grouped as follows:

- Timing Analysis
  - Time-driven Analysis
  - Access-driven Analysis
  - Trace-driven Analysis
- Power Analysis (PA)
  - Simple Power Analysis (SPA)
  - Differential Power Analysis (DPA)
- Electromagnetic Analysis (EMA)
  - Simple Electromagnetic Analysis (SEMA)
  - Differential Electromagnetic Analysis (DEMA)

- Optical and Optical Beam Based Analysis
  - Photon Emission Microscopy(PEM)
  - Thermal Laser Stimulation (TLS)
  - Electro-optical Frequency Mapping (EOFM)
  - Laser Logic State Imaging (LLSI)

In the following paragraphs, we will provide more details and insights on the different attack vectors and outline potential protection and mitigation approaches.

Timing Attacks

Timing analysis is an SCA that is used to extract critical information about the device under attack by analyzing the execution time of each operation under different setups and input patterns.

Attack Vectors. An adversary often applies timing analysis on cryptographic systems to extract the secret key, where timing analysis can help the attacker determine which subsets of the key are correct, and which subsets are not. Timing attacks are usually applied along with other side-channel attacks, since more information can be extracted when different analysis methods are employed. Power analysis is one example that works well with timing attacks; the power trace does not only show the pattern in which the operation performed is correlated to, but also how long it took before the operation is completed. The order of operation is also revealed when applying timing analysis to power signals; this order can help identify the type of process the device is running, and may even allow the adversary to reverse engineer the device.

Protection Mechanisms. To protect devices against timing attacks, designers can do the following: (1) randomize the delay of different operations, or (2) make all operations take the same time, thus preventing information leakage through timing channel. While constant-time implementations can guarantee security against timing attacks, they are not easily achievable in practice when applied at software level. However, during hardware design, full control over operation and timing behavior can help to avoid non-constant execution and significantly decrease success rates of such attacks. Randomization on the other hand, for instance, by adding random delays to the execution of a task, is easier to accomplish. While it makes the attack more difficult, it cannot, however, guarantee the security of an implementation against timing attacks. Randomization is done by creating various execution paths and adding different delays to different paths. One way to apply delay to a path is to place a series of buffers in the path during circuit design, where the number of buffers can be controlled by the designer to maintain the desired delay.

Power Analysis Attacks

The basic idea of power analysis attacks is to reveal secret information from a device by analyzing its instantaneous power consumption.

Attack Vector. Power analysis attacks are non-invasive and require physical access to the device, due to the need to capture current signatures that are produced while the device is undergoing an operation. SPA is a technique that aims to observe power measurements obtained while the device under attack is in operation mode. This type of analysis does not require any advanced or statistical processing stages. Visual inspection of power traces is considered the primary form of SPA attack, where a power trace shows a sequence of patterns that can lead to identifying key bits, instructions, or functions. DPA attacks are the most common type of side-channel attacks, due to the fact that attackers are not required to have prior knowledge about the hardware architecture of the device under attack to perform the analysis. Additionally, DPA has been proven very effective in obtaining high-quality signals in a noisy environment. Compared to SPA, DPA typically requires larger number of traces, however, additional data collection makes DPA more powerful. DPA is widely used to reveal secret keys of cryptographic systems by obtaining power traces while the system is encrypting or decrypting data blocks.

Protection Mechanisms. In the course of time and research, three main directions of countermeasures emerged and were applied in practice. While hiding countermeasures (such as sense-amplified-based logic (SABL) [TV04], wave dynamic differential logic (WDDL) [TV05]) mainly aim to decrease the Signal-to-Noise (SNR) ratio in order to embed and hide the information leakage in physical and random noise, masking countermeasures (such as d-private logic circuit [ISW03]) tackle the information leakage through randomization of the sensitive information. In particular, masking is applied on an algorithmic level, hence independent of the underlying physical architecture and device characteristics, using techniques from the domain of secret sharing and multi-party computation. As a third category of protection mechanisms re-keying countermeasures usually are applied for key-based cryptographic algorithms and implementations, while frequently updating and exchanging the secret key and in this manner, limiting the information leakage of the sensitive information according to a predefined threshold.

Electromagnetic Analysis Attacks

EM SCA focuses on measuring electromagnetic waves that are emitted from ICs in operation.

Attack Vector. These EM waves are defined as synchronized oscillations of electric and magnetic fields that propagate at the speed of light through a vacuum. The EM waves are produced as current flows across a device, where transistor and interconnect switching activities occur with changing input patterns. An adversary usually aims to capture EM signals that are produced by current flows of data processing stages, where most waves occur, due to the switching activity of a device while performing a data processing operation. These waves are usually considered unintentional, and they allow critical information to be leaked naturally during operation. When applying EM side-channel analysis, switching activities can be easily captured and translated into a

series of events and instances that occur in each clock cycle. This type of attack is similar to the power side-channel analysis, where a one-dimensional view of current activity is used to extract critical secret from a device. Power analysis attacks, such as DPA, however, cannot extract any spatial information, e.g., the location of a specific current activity. On the other hand, an EM side-channel attack can also identify the location of an EM signal, which makes it a powerful attack vector. EM signals often propagate through conduction and radiation; these signals can be intercepted using sensors, such as a near-field probe or an antenna. Using these sensors allows the EM signal to be transferred into a current signal, which is post-processed to remove noise, and limit the frequency band in order to apply the EM analysis.

Protection Mechanisms. To protect against EM SCAs, many countermeasures have been introduced. Redesigning the circuit to reduce the coupling issue is one of the primary countermeasures. Additionally, adding a layer of shielding to the device to prevent EM signals from propagating is another significant measure. Introducing nonfunctional modules that produce EM noise can also prevent critical information from being easily intercepted due to the high amount of noise being applied in the same frequency band. Further, due to the strong relation of power and EM side channels, all common countermeasures and protection mechanisms against power analysis attacks (hiding, masking, re-keying) can also prevent leakage of sensitive information through EM side channels and hence can be considered as appropriate protection mechanisms.

Optical and optical beam based analyses

Optical as well as optical beam based analysis focuses on optical information that can exit the analyzed chip. For decades, the increasing number of metal layers has made an optical interaction with the device level of the chip (FEOL) impossible when the measurement is performed form the chip front side. Therefore, optical and optical beam based analyses are mainly performed from the chip back side. Here, the FEOL is covered by the bulk silicon which is transparent for infrared light.

Attack Vector. Photon emission microscopy is a passive optical side channel analysis that utilizes the phenomenon of photon emission effects in FET devices. When a standard CMOS circuitry is operated, its transistors are switching from one logical state to the other. These switching processes are the root cause for the chips power consumption and for a faint emission of photons. During the switching process the transistors traverse the saturation condition. This condition causes the transistors to emit photons. Using a highly sensitive photon emission setup the switching activities of an actively operated chip can be monitored, this does also include the reading of memory cells. The integration of several operation runs is a valid method to increase the measured signal intensity.
Optical beam based attack methods like Thermal laser stimulation (TLS), Electro-optical Frequency Mapping (EOFM) or Laser Logic State Imaging (LLSI) are based on an interaction of the chip with an external inserted laser beam. Hence, these

techniques are not a side channel analysis in the classical definition and a further description of these techniques is not given here.

Protection Mechanisms. The most generic protection mechanism against optical and optical beam based analyses is to prevent unwanted optical signals from entering or leaving the chip. Throughout the last decade several approaches tried to apply encapsulation schemes. To our knowledge, none of these left the state of prototyping. Even if the level of protection was adequate, scaling the method into a high volume production would not be possible or too expansive.
A dedicated protection mechanism against photon emission microscopy side channel analysis is widely adopted into commercial products. The faint photon emission intensity that is radiated from a CMOS in operation conditions requires to measure the system several times to bring the signal to noise ratio into a measurable range. This attack vector was mitigated by limiting the maximum number of repetitions of similar operations.

3.3.2 Side-Channel Resistance in Hardware

In this section, additional details and insights on side-channel protection and mitigation techniques are provided. In particular, as the main scope of this project is protection of hardware designs and cryptographic accelerators for RISC-V processors, we will focus the discussion on recent trends and research directions for side-channel protection mechanisms in hardware.

Mitigating Timing Attacks in Hardware

In general, critical timing behavior that can be exploited by adversaries can occur on both algorithmic level or within the physical hardware. However, timing differences on algorithmic level often result from switching and branching on sensitive data and as such are mostly a problem in software implementations running on a general purpose Central Processing Unit (CPU). More specifically, in physical hardware, switching or branching is realized and implemented as multiplexers that inherently have a constant runtime and usually do not introduce critical timing differences. Given this, any algorithmic timing differences and branching on sensitive data can be easily resolved in physical hardware implementations due to the inherent parallelism and static behavior of digital logic circuits.

In contrast to this, timing differences on circuit and physical hardware level are more concerning and hence a critical aspect that needs consideration when establishing side-channel resistance. More precisely, on circuit level, different signal and gate delays can result in unintentional parasitic effects. However, while these parasitic effects are well-known and understood and controlled from a functional perspective, the impact of such effects with respect to side-channel resistance and security is critical as the signal and gate delays may impact the security of protection mechanisms. In a worst-case scenario, timing differences on signal and gate level can leak secret and sensitive information even in the presence of appropriate protection mechanisms, e.g.,

against PA or EMA (for a more detailed discussion on this topic, see the next sections below).

Mitigating Power & Electromagnetic Attacks in Hardware

In contrast to protecting software implementations against SCA, hardware implementations face several challenges arising from physical properties and parasitic effects of modern digital logic, particularly based on CMOS technology. For this, the following section extends the discussion of major approaches to mitigate power and electromagnetic attacks in hardware and lists prominent examples and solutions for each category as well as major challenges and difficulties to realize appropriate SCA countermeasures in hardware.

Hiding Techniques. Hiding is generally known as a common class of countermeasures to protect cryptographic devices against SCA. A subset of hiding countermeasures, focusing on implementation in hardware, aims at equalizing the power consumption (i.e., reducing the Signal-to-Noise Ratio for side-channel leakage signals) to render the consumption independent of the processed data and sensitive information, mainly thwarting DPA attacks. These countermeasures, commonly known as DPA-resistant logic styles, usually implement the concepts of Dual-Rail Precharge (DRP) logic. Prominent examples for such logic styles include (without claim to completeness):

- Dual-Rail Random Switching Logic (DRSL) [CZ06]
- Masked Dual-Rail Precharge Logic (MDPL) [PM05]
- Sense-Amplified-Based Logic (SABL) [TV04]
- Wave Dynamic Differential Logic (WDDL) [TV05]

Modern hiding techniques based on power equalization and custom logic styles usually have to overcome three major challenges. At first, early propagation [SS06] is related to unintentional switching of gates due to different delays of input signals arriving at the gate. In essence, for some logic styles, gates will evaluate at different points in time, highly depending on the input values and delays, hence leaking input information through timing differences. The second phenomenon and challenge faced in hardware devices are glitches [MS06], likely occurring at gate outputs if gate input signals will change during the evaluation phase of the logic style. For this, any DRP scheme has to ensure that the evaluation phase is only initiated after all input signals have become stable, otherwise, sensitive information can be leaked through any transient but unintentional computation and evaluation. Lastly, imbalanced routing [Tir+05] is the third major challenge for modern DPA-resistant logic style. More precisely, routes of different lengths will have different capacitive loads, hence having different contributions to the amount of power consumed on a signal toggle. As a consequence, any DPA-resistant logic style must strive to balance all related routes to minimize corresponding data-dependent leakage.

Masking Techniques. Among all countermeasures in hardware, masking (based on concepts of secret sharing) is one of the most promising countermeasures against SCA due to its formal and sound security foundation. However, as mentioned before, implementation of masking schemes in hardware is a non-trivial task, mostly due to parasitic effects [Fau+18] such as combinational recombinations (glitches), memory recombinations (transitions), and routing recombinations (couplings).

For this, Threshold Implementations (TIs) have been proposed as a first approach to address the issues and effects of glitches in hardware masking schemes [NRR06]. In particular, TIs rely on the properties of non-completeness and uniformity for masked logic circuits to ensure secure computation and combination of masked circuits in the presence of SCA adversaries. Further, many TIs can be applied already at an algorithmic level, hence providing a countermeasure that is independent of physical characteristics and properties of the final target device.

Nearly at the same time of the introduction of TIs, an entirely new branch of research started to focus on development of formal models for adversaries and physical execution environments to simplify and assist designers in implementation and formal verification of masking schemes in hardware. For this, in the context of masking, formal verification is often conducted in the simple, abstract, and elegant Ishai-Sahai-Wagner (ISW) d-probing security model [ISW03] (under some basic assumptions on noise distributions and independence of inputs), which allows an adversary to probe (observe) up to d intermediate values during the processing of sensitive information.

Given such a formal verification model, many different hardware masking schemes have been proposed until now, each providing different trade-offs with respect to circuit area, computational latency, and demand for fresh randomness during circuit evaluation to maintain security in the presence of a d-probing adversary. Among all proposed and introduced candidates, we would like to mention and reference a short selection (without any claim to completeness):

- Consolidating Masking Schemes (CMS) [Rep+15]
- Domain-Oriented Masking (DOM) [GMK17]
- Unified Masking Approach (UMA) [GM18]
- Generic Low-Latency Masking (GLLM) [GIB18]
- Hardware Private Circuits (HPC) [Cas+20]

However, securely masking arbitrary hardware circuits and designs is a non-trivial task, and complexity increases with design size. For this, the concept of secure gadgets was introduced, allowing to create and mask atomic components and functions that can be re-used and composed to create arbitrarily sized masked circuits. Unfortunately, creation and composition of d-probing secure circuits does not necessarily result in a d-probing secure circuit again, but might even reduce the SCA security during the composition process. For this, new security notions, allowing to capture and express the secure composition of gadgets, under some basic assumptions and requirements, were introduced.

For this reason, the following additional security notions have been introduced to extend the verification process and reason about the composability of secure masked gadgets:

Non-Interference (NI) [Bar+15] allows partial access through probes ensuring indistinguishability from circuit simulation.

Strong Non-Interference (SNI) [Bar+16] corrects deficiencies in NI through limiting information on output probes.

Probe-Isolating Non-Interference (PINI) [CS20] limits information propagation in share domains for trivial composition of secure gadgets.

In the presence of such composition strategies and appropriate (secure) gadgets, complexity of constructing arbitrary masked circuits is reduced through bottom-up construction. However, as each secure gadget usually comes at overhead in terms of circuit area, computational latency, and demand in fresh entropy, following such construction strategies usually ends up in larger and more inefficient circuits than creating masked circuits in a top-down approach (starting to integrate masking countermeasures already at the algorithmic level).

Mitigating Optical Attacks in Hardware

In U.S. Pat. No. 7005733, an anti-tamper structure is disclosed. In this structure, a light source is positioned on the surface of the IC and several sensors on the IC are used to detect the reflected light. This structure has the drawback that it restricts the application range of the IC as the IC must be surrounded by a transparent encapsulant which is enclosed by a reflective outer covering. Furthermore, because of this capsulation, heat cannot leave the package and the IC gets hot. Another tamper protection of the IC back side is disclosed in U.S. Pat. No 8198641. In this protection method, it is suggested to equip the chip back surface with a light-modifying structure such as lenses, large surface roughness, or reflective particles. The integrity of the back surface can then be checked by utilizing the IC structures to emit and detect light inside the IC. A drawback of this tamper-resistant structure is that the light source is a silicon p-n junction which is not an efficient light source and will degrade very fast. Another drawback of this method is that it is incapable of protecting the IC against attacks using optical techniques which are not harmful to the back surface. A next generation protection mechanism was presented by Amini et al. [Ami+18]. Here, an opaque protection layer is deposited on the back side of single chips that need to be protected. This technique provided a sufficient level of protection as well as the detection of harmful back side modifications. However, the presented method was only demonstrated on single chips so far. In the framework of this project, we will analyze if the protection mechanism can be transferred into a wafer level fab environment.

Protection of RISC-V against SCA

We would like to further investigate mitigation techniques against SCA for RISC-V processors that have been published in order to find the optimal solution as a

countermeasure. The aim is to protect the device against power or electromagnetic attacks while keeping the implementation costs as low as possible. Some RISC-V side channel attack works will be studied thoroughly [DGH19], [AKS21].

3.3.3 Hardware Trojans

Hardware Trojans are malicious modifications or added circuitry to exploit hardware or to use hardware mechanisms to leak secret information. It is extremely hard to detect hardware trojans due to the ever-decreasing feature sizes and high complexity of IP blocks. Moreover, trojans are designed to be activated under special conditions such as a specific power or temperature, making the detection harder even if high fault coverage tests are used. In order to assess the possible hardware trojan insertion points in the design flow, one needs to examine each design step. Each step is explained below.

- Chip Specification: The design flow starts with the chip specification, in which an engineer defines features, functionalities, and specifications. Two different teams, design and verification teams are involved in this step.
- Design entry/Functional verification: Here, logical behavior is confirmed by simulation. The design team and verification team generate Register-transfer level (RTL) code using test-benches. This is known as behavioral simulation. The functional blocks are also known as IPs (Intellectual Properties). RTL level IPs are called soft IPs and are either developed in-house or by third parties.
- RTL block synthesis: Once the RTL code and test-bench are generated, the RTL team translates the RTL code into a gate-level netlist using a logical synthesis tool that meets desired timing constraints. Once again, gate-level IPs (firm IPs) can be either designed in-house or procured from a vendor. After that, logical equivalence check (LVC) ensures functional compatibility between RTL code and the netlist.
- Chip partitioning: The engineers partition the entire design into multiple functional blocks (hierarchical modules). Once all the functional blocks are implemented in the architectural document, the engineers need to brainstorm design partitioning by reusing IPs from previous projects and providing them from other parties.
- Design for Test Insertion: When timing constraints are met with the logic synthesis, the design proceeds to the design for test (DFT) insertion step which includes scan path or memory BIST (Built-in self-test) insertion. In many cases, it is mostly outsourced to third-party vendors.
- Floor-planning: Physical implementation (RTL-to-GDSII) process starts with floor planning, where blocks and pins are placed in the chip. The aim is to minimize the chip area, make the chip easily routable and improve signal delays. It is also possible to integrate hard IPs from other vendors and integrate them into physical design.

- Placement: Standard cells are placed in a row in this stage. Once again, the aim is to have an optimum area while meeting the desired timing requirements. The connection of cells and the net lengths should be taken into account.

- Clock tree synthesis: In this stage, the clock-tree is designed to meet the specified timing performance.

- Routing: The routing step adds wires needed to properly connect the placed components while obeying all design rules for the IC.

- Final Verification: The design layout undergoes three different physical verification. Layout versus Schematic (LVS) checks if the layout matches the schematic (netlist). Design rule check (DRC) checks if the layout follows all the geometrical rules that are given by the manufacturing foundry. Lastly, logical equivalence checks (LEC) is the process of equivalence check between pre- and post-design layout. For example, microprocessor designers use equivalence checking to compare the functions specified for the ISA with a RTL implementation, ensuring that any program executed on both models will result in an identical update of the main memory content. At every stage, we need to make sure that the logical functionality is intact and does not break because of any of the automated or manual changes. This is why LEC is one of the most important checks in the entire chip design process.

- Graphical Data Stream Information Interchange (GDSII): GDSII file is required by the manufacturing foundry for the fabrication. It is used to reconstruct or transfer the layout between different tools and to create the photomasks. GDSII files are usually the final output product of the IC design cycle and are handed over to IC foundries for IC fabrication.

Potential Adversaries. The globalization of chip design and fabrication poses a big threat to the security of integrated circuits. As discussed above in the design flow, fabrication of a chip from beginning to the end mostly requires the involvement of different parties. The potential adversaries might be third-party vendors, DFT-insertion vendors and manufacturing foundries. As stated above, the design house might need to outsource in order to speed up the production. Soft, firm and hard IPs might be procured from a third-party vendor, which is a possible insertion point for a hardware trojan. Since the RTL code of complex designs is extremely long, it is very hard to detect any added code as a malicious attack. Similarly, it would be extremely hard to detect any added circuitry in the layout since today's chip design has billions of transistors. Alike, DFT vendors have full access to the design and they add their custom design DFT circuitry, which might have malicious circuitry. Another hardware trojan insertion phase is during the tape out. The third-party foundry has full access to the design. This step is the most vulnerable since it is very hard to detect.

Hardware Trojan Design. Hardware trojans are malicious modifications of a chip by an adversary that can disable the circuit, change its functionality, or create a hidden

side channel through which a secret can be leaked. They can be implemented by adding/removing logic gates during the design/fabrication or by changing the physical parameters of existing logic during manufacturing. A trojan design might include two main parts, trigger and payload [BT19]. The activation mechanism is referred to as trigger, and the part of the circuit or the functionality affected by the activation of the hardware trojan is referred to as payload. Once the trigger is activated, which is deliberately designed to be very rare, the payload performs malicious behavior. In case the payload is not activated, the chip behaves like a trojan-free circuit. An ideal trojan should not be detected during the testing phase. Trojan circuitry might be combinational or sequential. They can also be classified as digital or analog based on their signal source. Digital Trojans can either affect the logic values at chosen internal payload nodes or modify memory locations' contents. Analog payload Trojans, on the other hand, affect circuit parameters such as performance, aging power, and noise margin. The attacks mentioned above are introduced into the system during the circuit design phase via inserting codes into RTL. Another insertion point in an IC production is the fabrication phase, where a malicious adversary can change layout design or mask. An adversary might simply change the process parameters or the wiring. The hardware trojan insertion points in the value chain are described below.

Hardware Trojan Insertion Points in the Value Chain. As a chip goes through different phases of manufacturing, it might be subject to malicious attacks. These phases are described below.

- Specification: As a first step in the design below, the system characteristics are defined in this phase. The factors to be considered in this phase include performance, functionality, and physical dimensions. The end results are specifications for the size, speed, power, and functionality of the system. In this phase, functional specifications can be altered by a malicious actor.

- Design: Designers might use third-party IP blocks and standard cells and trojans might be inserted into any of the these components, such as a standard cell library can be modified by an attacker.

- Fabrication: During this phase, developers create a mask set. A malicious attacker might make small changes on the mask that might cause serious effects. Moreover, process parameters or chemical compositions might be altered during fabrication to cause failures or speed up the aging.

- Testing: An attacker can modify the test pattern in a way that a hardware trojan inserted during design or fabrication will not be detected while testing.

- Assembly: Assembly is the phase during which a chip and other components are mounted on a printed circuit board (PCB). A malicious and untrusted assembly might create security flaws in the system intentionally. For example, a trojan can be inserted by introducing an I/O pin with high capacitance to affect the performance. It is also possible to replace a component with a malicious one in this phase.

3.3.4 Activity Induced Stress as Hardware Threat

Whether enforced by malicious intent, caused by non-ideal design, fluctuations during the fabrication or heavy usage, stress poses another threat to the hardware of a computational system.
In a broader sense, stress describes the physical strain that a system is exposed to at a given point in time, whose effects either directly or indirectly influence the service life of the system. This stress is caused by external influences such as temperature, humidity and vibrations or internal circumstances, for example expansion coefficients of the metal, voltages or field strengths.

Effects of Stress on Hardware

The effects of stress range from a gradual shift of the electrical parameters of a transistor (aging) to its thermal breakdown and are a field of continued research. Four of the main stress-induced defect causes can be summarized as follows [WB08]:

- Electromigration: Electromigration (EM) occurs when metal atoms are accelerated and consequently transported inside interconnecting wires. The atoms are migrated from regions with high current density to regions with low current density, resulting in increased resistances (thinner conductors) and short circuits (unwanted bridges).
- Hot Carrier Injection: According to More ([Mor11]), HCI occurs when a charge carrier is moved along the channel of a MOSFET near the side of the drain. There, it is accelerated by the electromagnetic field. If the kinetic energy created by this acceleration is greater than the energy that the crystal lattice contains in thermal equilibrium, they are called “hot” carriers. With help of this energy, the charge carrier itself can penetrate the oxide of the gate or (via impact ionization) shoot other carriers there and thereby alter the electric characteristics of the MOSFET.
- Time-Dependent Dielectric Breakdown: The exact physics of TDDB are still not fully understood. Most researchers support the belief that either the applied voltage itself or the resulting tunneling electrons create defects in the volume of the oxide film. Once these accumulating defects reach a critical density, a sudden loss of dielectric properties is triggered, which leads to a surge of current and a large localized rise in temperature, ultimately resulting in permanent structural damage within the silicon oxide film.
- Negative Bias Temperature Instability: This electrochemical reaction takes place in PFETs. Negative bias temperature instability (NBTI) leads to an increase in the threshold voltage in a transistor, which in turn causes errors due to violation of temporal conditions.

The ongoing trend of technology scaling only adds to the stress on the material and the resulting reduction of lifetime. Smaller physical dimensions of the circuits and conducting materials, strongly increased power density and the resulting rise in temperature are accelerators for all of these effects.

Stress Mitigation Techniques

There exists quite a number of approaches to counter the threats of stress and the resulting effects on hardware, but we will concentrate on the following:

- Lowering the power density: The obvious method to avoid stress is to lower the power density. Unfortunately, due to numerous reasons, there is a scaling gap between the physical feature sizes of the transistors and the applied voltages and clock frequencies ([Sri+04]). Therefore, reducing the power density is hardly possible for state of the art technology.

- Lowering the temperature: The in-die temperature has a direct influence on all of the aforementioned defect models. According to Viswanath et al ([Vis+00]) it is a well-known fact that the reliability of transistors is exponentially dependent on the operating temperature of the junction. Furthermore, the authors claim that a small difference in operating temperature of about 10-15°C already results in a ~2X difference in the lifespan of a system. Therefore, reducing the temperature is very crucial. Jayaseelan has written an excellent thesis on thermal management ([Jay09]) and groups the different methods into passive approaches like improving heat conductors or spreaders and active measures like activity migration or load balancing (which will be explained in the next bullet point).

- Balancing and shifting the switching activity: Activity Migration and Load Balancing are two very similar, but also very effective methods of active thermal management, which rely on some form of redundancy to work. They both steer the rate of the switching activity in certain components and thereby directly influence the production of heat. Activity Migration is the process of shifting the active processing from one functional unit to another and thereby providing time for the original one to cool down. Load balancing differs in evenly spreading the overall load over the available functional units in the first place and thereby avoiding the generation of so called hot spots. Even though quite similar, each method has it's advantages and disadvantages and might not be applicable in any case.

- Fault tolerance: In contrast to the other methods, fault tolerance is not aimed at avoiding stress or defects in the first place, but at mitigating the resulting consequences. This is achieved by using different forms of redundancy on all levels of abstraction of hard- and software and usually involves the steps of identifying faulty results or processes and partially also masking their occurrence. An excellent overview on this topic is given by Koren and Krishna ([KK10]).

4 Prototype Processor Architecture Requirements

For the development of an open source processor with an integrated cryptography hardware accelerator, a suitable Instruction Set Architecture (ISA) has to be selected. Using the ARM-ISA poses problems in so far as it is fully controlled by the ARM Limited company and licenses would need to be obtained for development and modification. This would negatively impact the benefits of an open source implementation as users would have to obtain a license first. The value proposition of the HSM demonstrator is heavily influenced by the licensing arrangements necessary for its modification and use. This means that having a permissive open license for all parts of the project is critical for its utility. A license-free access for universities will promote uses in teaching and make research results more easily compared. A permissive license will also have influence on the supply chain as suppliers can be changed easily, helping to avoid supply shortages and driving competition. Conversely, supply chains can be interrupted by licensing problems such as an international trading conflict leading to sanctions that prohibit further licensing.

A more recent alternative to the ARM-ISA, RISC-V, offers an open source compatible ISA that does not require licensing and is especially suited for extension by special purpose hardware. Additionally, there are many open source RISC-V implementations which could be used as base for the HSM.

4.1 The RISC-V ISA

The RISC-V ISA has a modular design with base instruction sets (for 16, 32, 64, 128 bit integer arithmetic and a load/store machine) and optional extensions. The following extensions can be used to enhance a given base instruction.

- `Multiplication:` The `M` extension introduces instructions for integer multiplication and division. The inclusion of this extension represents a time-area trade-off for the user as a hardware multiplication unit will consume more chip area but perform the required operation much faster. Many cryptographic schemes make heavy use of integer multiplication and would benefit from the use of this extension.

- `Compressed Instructions:` The `C` extension introduces short 16 bit instructions for common operations, thereby reducing the code size. This can often reduce the required memory by 25% [AW19]. The significant reduction in code size is attractive for space intensive firmware, but is only realized in the instruction memory (for example on-board FLASH memory).

- `Floating-Points:` The `F`, `D`, and `Q` extensions introduce IEEE-754 floating-points with 32(`F`), 64(`D`) or 128(`Q`) bit precision. Floating Point Units can be expected to consume a lot of chip area and are probably not required for the implementation of an HSM. If the performance of floating-point operations is not critical they can also be easily replaced by software implementation already present in open source compilers.

- `Atomic Instructions:` The `A` extension introduces atomic memory instructions that are useful for thread safety and lock-free programming.

- `Bit-Manipulation:` The `B` extension aims to introduce a variety of bit manipulation instructions that speed up otherwise complex operations like counting zero bits in a word. This can be useful for implementing symmetric cryptographic primitives [Mar+21] such as ciphers and hash functions. The extension has not been adopted by RISC-V ISA at the time of writing. Once adopted, the ISA will probably include a range of bit manipulation extensions, with the `B` extension itself being a selection of different instructions from those extensions.

The notation for RISC-V processors is the name of the base instruction set followed by a set of letters that identifies the extensions used. `RV32IM`, `RV32EMC` or `RV32IMC` appear to be the most suitable bases for the implementation of an HSM. From this, one can select several specific variants like rv32im (the 32bit base instruction set with integer operations and integer multiplication). A 32bit instruction set is indicated if any application uses more than `64kB` of memory. Experience shows that 32bit controllers are almost always used for new designs in the automotive industry. This will certainly also apply to future developments, which speaks in favor of an ISA that allows 32bit implementations and simple extension to 64bit. While a 64bit architecture is possible in general, the required chip area increases significantly. As the larger address space is not needed, a 32bit architecture is to be preferred.

4.2 HSM Prototype specific Requirements

The HSM demonstrator will have hardware-specific requirements that result primarily from the HSM-specific use-case requirements described in Section 3.2 and the demands of the HSM firmware that is used to implement these use-cases. In the following, these requirements are discussed in their respective context.

- `ISA:` The HSM Firmware will not have use for IEEE–754 floating point number arithmetic and it is unlikely that atomic operations will be required. However, software implementations of public key cryptography make extensive use of integer multiplication. If no hardware multiplicator is present, as is the case in the base instruction set, integer multiplication is realized by the compiler using the available integer manipulation instructions. This has serious implications for performance of the cryptographic primitive that can be estimated as two orders of magnitude. Furthermore, integer multiplication is often used with

secret values and has to run in constant time to avoid timing side-channels. This holds true for all instructions, and software in general should avoid using variable time instructions with secret values as much as possible, if at all. Hardware multiplication structures with the appropriate performance and constant time behavior use a lot of chip area. It would be possible to use the base instruction set if a dedicated hardware accelerator for the required schemes is implemented. Therefore, the HSM requires either `RV32IM` or `RV32I` with appropriate hardware accelerators. Additionally, compressed instructions (provided by the `C` extension) could prove very useful if significant savings in memory can be realized.

- `Memory:` The memory requirements of the HSM firmware are made up of two distinct quantities. The first is memory that is used to store the program code of the HSM firmware itself, and the second is the memory the firmware needs for various computations and to keep the internal state and data at run-time. The program code for an embedded system is usually stored in non-volatile external memory and is connected to the processor with an appropriate interface. The common implementation of this is FLASH memory that is attached using the SPI bus. Memory for data and state is usually implemented using SRAM inside the processor. The exact quantities that are required are very hard to determine in the absence of a complete implementation and are subject to time-memory trade-offs and optimization during implementation. We estimate that the HSM will require on the order of `100 kB` of data memory and `200 kB` of program memory for a functional implementation. Since the program memory can be external, the only requirement for it is that an appropriate interface (`SPI`) is available. If memory caches are used, they might introduce timing side-channels into certain cryptographic software implementations. Use of these would have to be avoided or would require extensive knowledge of the exact caching behavior to avoid the accidental creation of side-channels during implementation.

- `Communication Interfaces:` A way for communicating with the HSM must be implemented for it to have any utility. Part of the TPM 2.0 standard is a simple memory-mapped communication interface [TIS] that could easily be adapted for the purposes of the HSM. One of the implementations of this interface uses `SPI` to implement read/write for this memory mapping. However, any low-level communication bus can probably be used to implement this. To aid the development process a `JTAG` and an additional `UART` interface are probably very useful, but they are not required for the functionality of the HSM firmware. With these somewhat optional interfaces and including the `SPI` bus for external memory mentioned above, the firmware requires a total of 4 communication interfaces. One satisfactory selection would be `SPI` (master), `SPI` (slave), `UART`, and `JTAG`, but many combinations are possible. A `JTAG` interface can be used to read and write all memory and control the processor very precisely, thus posing a security risk. Therefore, the implementation must be able to temporarily disable the `JTAG` interface to protect the confidentiality of sensible data in memory. These interfaces should be implemented in hardware to avoid performance problems that can arise from using interrupt driven software

implementations.

- `Cryptography:` Part of the HSM will be hardware structures for the acceleration of cryptography. This may include a true random number generator (RNG), a hash function primitive like SHA2 [SHA93], a symmetric cipher like AES [AES], and potentially accelerators for public key cryptography like RSA [RSA] or elliptic-curve cryptography [ECC]. For communication with this additional hardware, a method for attaching co-processors is required.

- `Trust Anchor and Secure Storage:` Implementation of a tamper-protected secure storage and a trust anchor as specified in Section 3.2 requires that the HSM incorporates a short program that is always executed before the firmware itself is executed. This bootloader has the purpose of examining the firmware for tampering and preventing access to the secure storage if tampering is detected. For these reasons, a read-only memory (ROM) or internal non-volatile memory is required as well as a mechanism for adding either to the processor. If no internal non-volatile storage is available, there is no trivial way to implement a protection against replay attacks or to implement rate-limiting mechanisms in the firmware.

4.3 Open Source RISC-V Implementations

There are several RISC-V implementations that are available under an open source license. The choice of implementation influences and is influenced by the requirements for development tools listed in Chapter 5. For the construction of the demonstrator we are primarily interested in the different mechanisms for adding special-purpose hardware structures to the core in question. A selection of interesting candidate RISC-V implementations is given in the following:

- `Picorv32:` The addition of tightly coupled co-processors is possible via a custom interface that allows non-branching instructions to be implemented on a co-processor. This interface is used to implement the `M` extension for this implementation, but it could just as well be used to interface with special purpose hardware for the acceleration of cryptographic operations. It would be theoretically possible to use Verilog-producing HDLs to implement such a co-processor as long as the result respects this co-processor's interface. The Picorv32 uses the ISC open source license that is equivalent to the BSD license; hence it is very suitable for use in an open source project. The implementation, however, is optimized for size, with binary shifts taking up to 14 cycles and multiplication operations taking up to 72 cycles [1]. This might prove suboptimal for software implementations of cryptography and require extreme scrutiny to avoid the accidental creation of timing side-channels. The Picorv32 can also implement the `C` extension for compressed instructions.

[1] `https://github.com/cliffordwolf/picorv32#features-and-typical-applications`

- `VexRiscv:` The VexRiscv is is a pipelined RISC-V processor implemented in SpinalHDL. The very generic Pipeline Component can be extended by plugins that implement any subset of the RV32I/E[M][A][F][D][C] extensions. In many cases there are multiple plugins for the same functionality. The SpinalHDL implementation generates Verilog output and can easily be used with Verilog based tooling. VexRiscv uses the MIT Open Source License which is very permissive to use, especially if the goal is to create another open source product. The modular design makes it easy to extend the 5-stage pipeline with custom instructions and tightly coupled special purpose hardware. The design also allows to easily have co-processors that have direct memory access. There exists support for generating modified Verilog output that includes annotation for formal verification tools. Simulation is easily accomplished with Verilator and SpinalHDL derived test-benches, and further testing with FPGAs is entirely possible with open source tools.

- `Rocket Core:` Similar to SpinalHDL, Chisel is based on Scala and creates FIRRTL and Verilog (therefore it is useful for open source synthesis tools). Also similar to the VexRiscv, the Rocket Core is also a 5 stage pipeline. With most of the code being licensed under the BSD license the Rocket Core could easily be used as basis for an open source project. The Rocket Core is a proven design and has been used to implement the first commercially available RISC-V microcontrollers. The Rocket Core can implement subsets of `RV32IMA`.

- `PULPissimo:` PULPissimo is a pipelined RV32 processor with 4 stages and an interface for co-processors that have memory access and can implement subsets of RV32I/[M][F][C]. The peripheral interface includes a direct memory access (DMA) layer allowing for implementation of peripherals that can directly access memory without additional delays that would occur if the processor was to move the working data to and from the co-processor. This allows for the easy implementation of high performance hardware accelerators for various cryptographic operations. For example, hash function accelerators could make use of this. However, SystemVerilog might restrict the choice of open source synthesis tools or make it impossible.

Rather than selecting a specific implementation and thereby committing to a Hardware Description Language, the merits of these different implementations shall serve as decision aid in the selection of the EDA tools in Chapter 5. However, the availability of an open source RV32I implementation that can be adequately extended is highly desirable for the implementation of the demonstrator. This holds true for all the examples above, to some extend.

5 Requirement Definition and Choice of Development Tools

For the different development tasks in assembling a processor and firmware that operates it a set of tools is required. In the scope of this project we will design and build an HSM to demonstrate the ultility of those tools. The central component of the HSM will be a processor running a firmware that implements a subset of the TPM2.0 specification. The required hardware synthesis needs to support the development flow for ASIC and FPGA to allow for proper testing and evaluation before the actual fabrication. Furthermore, the simulation of circuits needs to be supported for the development of large circuits. A hardware description language (HDL) has to be selected that supports these features as well as formal verification. Alternatively, the HDL has to be extended to support the required features. The software components of the demonstrator will probably be written in C or Rust and will require a compiler that supports the RISC-V target. With the aim of developing an open source processor with an open source firmware, open source tools should be preferred here. This would also ease the modification of these tools to satisfy requirements that could not be met previously. With this in mind we arrive at the following functional and non-functional requirements:

Functional Requirements	Non-functional Requirements
Synthesis	Open Source
Placer	Interoperability
Router	High Level of Abstraction
Timing Analysis	Simulation
ASIC flow	Formal Verification
FPGA flow	
C Compiler	Debugger
RISC-V support	

5.1 Development Tools

The development of hardware requires the decision on several different tools such as a suitable Hardware Description Language (HDL), a simulator, a synthesis tool and a place&route tool. Multiple factors play a role in finding the most suited tools, such as the available functionalities, but also whether the tool is open-source or proprietary, with an open-source tool being the preferable solution. Figure 5.1 gives an overview on existing open-source tools in the hardware development tool flow.

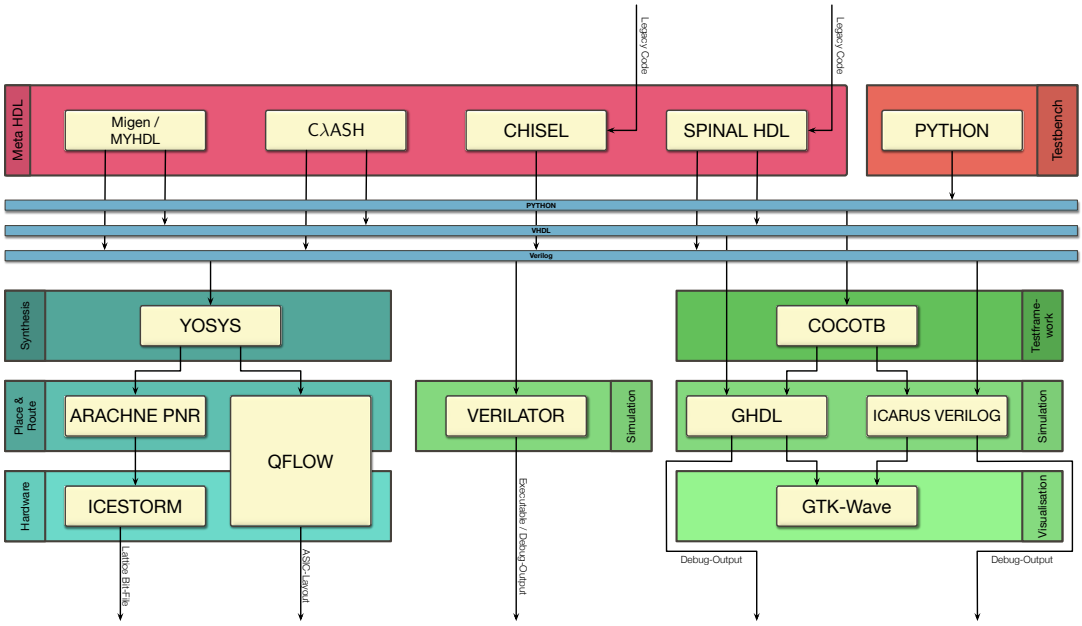

Figure 5.1: Open Source Hardware Development Tools

5.1.1 Languages

Apart from the well-known HDLs VHDL and Verilog, newer language are often domain-specific languages embedded into higher-level languages such as Scala or Python. These languages, sometimes referred to as Hardware Construction Languages, still allow the developer to take full control over the created hardware while introducing features from functional and object-oriented programming. The approach of High-Level Synthesis (HLS) takes the programming to an even higher abstraction level, allowing the developer to write software code and then have it translated into HDL code by a tool. The selection of an appropriate HDL has applications for the Selection Criteria discussed in Chapter 4, in so far as the selection of an HDL will restrict the available open source processors that can be used as basis for a demonstrator. Conversely the availability of an open source RISC-V processor is a strong argument for the use of a given HDL.

Hardware Description Languages

- Verilog and VHDL are the most commonly used HDLs. As a consequence, most simulation and synthesis tools work with these two languages. The developer has full control over the resulting hardware, but the low abstraction level of the languages results in long codes that are harder to read and maintain, and it makes it difficult to create generic designs.

- SystemVerilog is a superset of Verilog with enhancements to address the system-level design and verification. It features multiple improvements over plain Verilog,

however, it would make the use of open-source synthesis tools difficult.

Hardware Construction Languages

- SpinalHDL is based on the functional object-oriented programming language Scala, resulting in a higher abstraction level. This allows for the creation of more generic designs while also requiring less written code. From the SpinalHDL code, equivalent VHDL or Verilog code can be automatically generated. The SpinalHDL community is very active with frequent updates to the language. As we are in contact with the main developer of SpinalHDL, the extension of the language and the addition of new features needed by us should be a lot easier than for other HDLs. Powerful libraries allow to extend the functionality of Spinal.

- Chisel is the precursor of SpinalHDL, with a less active developer community. Just like Spinal, Chisel is based on Scala and shares many of the positive aspects of Spinal. The first commercially available RISC-V processors, developed and sold by SiFive[1], were created using Chisel.

- Clash is based on the functional programming language Haskell, but extends it with dependent types to allow the strongly typed construction of hardware. The Clash compiler transforms these high-level descriptions to low-level synthesizable VHDL or Verilog. It emphasizes synchronous circuits.

- MyHDL is based on Python. Similarly to the Scala-based HDLs, it has a high abstraction level and generates Verilog and VHDL code. Its functionalities can be extended by powerful libraries, however, the project seems to be no longer active with the latest release being from 2018.

- nMigen is another Python-based language that shares the positive aspects from MyHDL. It seems to be more active with the latest commit to the Git being from May 2021. However, the language is still incomplete as the standard library and build system will undergo changes before the design is finalized.

- SystemC is a class of `C++` libraries interfacing an event-driven simulation kernel. It allows to model the system at transactional level. The user constructs a virtual prototype of the hardware in software using SystemC, allowing early software development. Only a small subset of SystemC is synthesizable. SystemC ist a standardised as IEEE Standard 1666.

High-Level Synthesis

High-Level Synthesis relies on powerful, often proprietary, tools to transform high-level software code into equivalent HDL code. The source languages are usually `C` and `C++`, however there exist a few tools that can process other languages. The advantage of HLS lies in the very high level of abstraction and the potentially quick design process,

[1] `https://www.sifive.com`

however, the developer has little influence on the generated hardware and has to rely on the HLS-tool to create the corresponding hardware. The list below shows some of the available HLS-tools.

- Vivado HLS is a proprietary tool developed by Xilinx. It can process inputs written in C, C++ or SystemC and outputs VHDL or Verilog code.
- Stratus HLS is a proprietary tool by Cadence that supports the same inputs as Vivado HLS and outputs RTL code.
- The Intel High Level Synthesis Compiler is part of the proprietary Intel Quartus Prime design software and allows to process C or C++ into Verilog.
- Catapult is a proprietary tool developed by Siemens that processes C, C++ and SystemC into VHDL or Verilog.
- Bambu by PoliMi is an academic HLS tool that can process C into Verilog. Unlike the previous proprietary tools, it does not support fixed-point arithmetic.
- DWARV is an academic HLS compiler developed at the Delft University of Technology. It can process a subset of C into VHDL.

5.1.2 Simulators

Simulators are used to simulate the behavior of the hardware that was specified using a HDL. The simulation allows to find potential errors in the design at an early stage. Simulators can generally be divided into proprietary and open-source tools. Some of the existing simulators are listed below and Table 5.1 contains a summary of their respective properties.

Proprietary Simulators

- ModelSim is a well-known and well-supported simulator developed by Mentor Graphics. It is capable of simulating separated or mixed VHDL and Verilog entities and supports a wide range of different standards of these languages. To the day, ModelSim is the leading simulator for FPGA design. Licenses are expensive, and the existing free version only allows to simulate smaller designs and runs at only 40% of the full version's speed.
- VCS is a well-established proprietary simulator by Synopsys that supports different language standards of VHDL, Verilog and SystemVerilog. It features multiple advanced simulation techniques such as native low power, x-propagation and fine-grained parallelism.
- Xcelium is developed by Cadence and supports a wide range of different language standards of VHDL, Verilog and SystemVerilog. It allows multi-core simulation for a reduced simulation time as well as several advanced features such as x-propagation.

Open-Source Simulators

- Verilator supports Verilog, but not VHDL. It typically has a high performance that matches that of proprietary simulators, and the possibility of multi-threading could further improve the performance of Verilator. Verilator has a wide range of functionalities, but as an open-source tool, it might lack some features of the proprietary simulators.
- IcarusVerilog is a light-weight open-source simulator that only supports Verilog.
- GHDL is a fast and open-source simulator that supports the simulation of VHDL.

	Open Source	Performance	Verilog
ModelSim	x	✓	✓
VCS	x	✓	✓
Xcelium	x	✓	✓
Verilator	✓	✓	✓
IcarusVerilog	✓	x	✓
GHDL	✓	x	x

Table 5.1: Summary of Hardware Simulation Tools

5.1.3 Synthesis Tools

Some of the existing proprietary and open-source synthesis tools are listed below and Table 5.2 contains a summary of their respective properties.

Proprietary Synthesis Tools

- The Vivado Design Suite by Xilinx is a set of tools that allows the simulation of the design via the built-in simulator XSIM, synthesis of both VHDL and Verilog, and place&route. However, licenses are expensive.
- Design Compiler Ultra is a synthesis tool developed by Synopsys that allows the synthesis of VHDL, Verilog and SystemC. The tool features a variety of optimization techniques that help it to improve the synthesis results.
- Genus Synthesis Solution is developed by Cadence and supports VHDL, Verilog and SystemC. Several optimization features improve the results of the synthesis.

Open-Source Synthesis Tools

- Yosys is an open-source synthesis tool that only supports Verilog. It works very well together with the place&route tool nextpnr and features ASIC and FPGA support via the icestorm project.

- The open-source Verilog to Routing tool flow consists of multiple tools. It takes as input a Verilog description of the digital circuit and a description of the target FPGA architecture, and outputs the FPGA bitstream. The synthesis is done with ODIN II.

	Open Source	Verilog	FPGA targets	ASIC targets	Formal Verification
Vivado	x	✓	✓	✓	x
Design Compiler Ultra	x	✓	✓	✓	x
Genus Synthesis Solution	x	✓	✓	✓	x
Yosys	✓	✓	✓	✓	✓
Verilog to Routing	✓	✓	✓	✓	✓

Table 5.2: Summary of Hardware Synthesis Tools

5.1.4 Place and Route

Some of the existing proprietary and open-source tools for Place and Route are listed below and an overview of their properties can be found in Table 5.3.

Proprietary Tools

- Innovus Implementation System is a place&route tool developed by Cadence which features multiple optimization techniques that help improve the timing and area results of the implementation.
- The Vivado Design Suite includes a tool for placement and routing.
- IC Compiler II is a proprietary place&route tool developed by Synopsys.

Open-Source Tools

- The tool arachnepnr is not maintained anymore, with nextpnr being a complete functional replacement with several major improvements.
- nextpnr is the successor to arachnepnr and works very well together with the Yosys synthesis tool.
- In the second part of the Verilog to Routing tool flow, the hardware description is packed, placed and routed using the Versatile Place and Route (VPR) tool.

In the medium term, it appears that there will be several alternatives for Place and Route. A promising future candidate is Luna[2], which is intended for IC processes with features sizes > 100nm. The authors announce that Luna will be released under a liberal open source license. Moreover, by using KLayout, a chip mask layout can be viewed and edited, further supporting the Place and Route.

[2] https://www.asicsforthemasses.com/

	Open Source	FPGA	ASIC	actively maintained
Innovus	x	✓	✓	✓
Vivado	x	✓	✓	✓
IC Compiler II	x	✓	✓	✓
arachnepnr	✓	✓	✓	x
nextpnr	✓	✓	✓	✓
VPR	✓	✓	✓	✓

Table 5.3: Summary of Place and Route Tools

5.1.5 Formal Hardware Verification

The used formal verification tools have to provide the necessary functionalities, enabling the user to properly formulate the desired properties for the design specifications and to incorporate certain solver engines for a bounded model-check or even a complete proof. Apart from that, there are a few more requirements concerning the whole design environment. To exploit the full potential of formal methods, they have to be embedded deeply into the design process. Particularly when using high-level HDLs like SpinalHDL or Chisel, ways have to be found to integrate formal properties and to correctly propagate them to the lower abstraction levels of the design flow. Some of the existing proprietary and open-source hardware verification tools and verification apps are listed below and Table 5.4 contains a summary of their respective properties.

Proprietary Formal Hardware Verification Tools

- Onespin 360 DV is a formal verification platform enabling the user to verify a design from multiple angles. There is a library of different verification apps, including one specifically testing for a gap-free compliance with the RISC-V ISA.
- RISC-V ISA Formal Proof Kit® contains a precompiled library of System Verilog Assertions to check any RTL design for compliance with the RISC-V ISA. There is also a verification app called formalISA®, which is built on top of the Proof Kit.

Open-Source Formal Hardware Verification Tools

- SymbiYosys is a frontend driver extending the Yosys toolchain and allowing for bounded/unbounded verification of safety/liveness properties, as well as trace generation from cover statements. It takes Verilog/SystemVerilog code with constraints as an input and utilizes a wide range of different solver engines to prove or disprove the statements contained therein. Regarding RISC-V specific verification, approaches like the open source framework ”RISC-V Formal” can be used for an end-to-end blackbox verification of the complete design.
- Kami-Framework is a Library for the Coq proof-assistant. It can be used to specify, implement and verify a design on a high abstraction level and then

automatically generate hardware components, following the style of the Bluespec language. Regarding RISC-V specific verification, multiple formal specifications of the RISC-V ISA are implemented in Kami and a library of already proven modules is available.

	Open Source	Compatibility with standard toolflows	RISC-V specific implementations
Onespin 360 DV	x	✓	✓
RISC-V ISA Formal Proof Kit®	x	✓	✓
Yosys/SymbiYosys	✓	✓	✓
Kami-Framework	✓	x	✓

Table 5.4: Summary of Hardware Verification Tools

5.2 Choice of Tools

The choice of a Hardware Description Language depends on multiple factors: We need a language with a high abstraction level that still allows us to have full control over the generated hardware. This eliminates low-level HDLs such as VHDL and Verilog, but also the high level synthesis approach, and leaves the Hardware Construction Languages. The most promising candidates are Chisel and SpinalHDL as they are very well-made with an active community, and there exist open-source processors written in these two languages (Rocket and VexRiscv). The VexRiscv has allready proven itself to be highly configurable and efficient is therefore the implementation of choice. As a consequence SpinalHDL will be used and in collaboration with the developer some vital formal verification functionalities have already been added.

Regarding the tools for hardware development, our available choices are very limited, as we want to rely on open-source tools wherever possible. The Verilator project seems to be the most suited simulator as it is open-source, fast, and supports Verilog, which can be generated from SpinalHDL. Our choice of a synthesis tool is Yosys because it offers the best functionalities among the open-source synthesis tools and has support for ASIC and FPGA targets. Working well together with Yosys and being open-source, nextpnr is the ideal place&route tool targeting FPGAs for us. Open-source place&route tools targeting ASICs are currently being evaluated, on the basis of those tools included in the OpenROAD project[3]. The following processing steps are determined by the semiconductor fab in respect to the booked production technology, in our case it will be a 120nm process. Nevertheless, it will be necessary to verify the functionality of the physically produced chip and its implemented tamper protection mechanisms.

For software development, there exists a plethora of tools that are open source and that can be used to implement the software components that are required. The HSM firmware can be implemented in a low-level programming language such as

[3]https://github.com/The-OpenROAD-Project/OpenROAD

C or Rust, and the existing open source tools can be used throughout the entire development process. This will most likely involve using the GNU Compiler Collection[4] which already supports the RISC-V ISA and its extension with custom instructions. Alternatives however exist, the LLVM project[5] also has support for the RISC-V ISA and is the basis for several C and Rust compilers.

[4]https://github.com/riscv/riscv-gnu-toolchain
[5]https://github.com/llvm/llvm-project

6 Definition of Formal Verification Methods

The main goal for formal verification in this project is to raise the quality of the development process, by ensuring correctness of crucial aspects of the design and thereby demonstrate the capability of open source solutions to do so. The Yosys/SymbiYosis-Toolchain fulfills all the necessary requirements to serve as a basis for the desired formal verification. The compatibility with SpinalHDL has already been ensured with the help of its designer, and therefore formal verification can be integrated in all stages of the design process. The potentially verifiable aspects can be grouped into two categories:

- Functional aspects of the main CPU and the HSM extension
- Non-functional aspects, which include countermeasures against different side-channel attacks or hardware-trojans

The formal verification process depends on the actual design and implementation. While a full formal verification covering all aspects would be beyond the scope of this project, we will focus on some key aspects of the development and demonstrate the general feasibility of formal verification for RISC-V systems. Moreover, due to the open source nature of the project, our verification efforts can serve as a blueprint and starting point for future verified RISC-V-based systems. Therefore the first step is to establish a modifiable and extendable method of verifying the main CPU‘s ISA-compliance on a higher level of abstraction. This way, the correctness of the core functionality can be guaranteed after changes or additions to the design. The riscv-formal framework presents a good starting point for this purpose; a bounded model check for a VexRiscv configuration already exists [1]. This framework will be evaluated and then adapted, also aiming for a more complete proof. In a second step, additional components of the architecture will be verified. In particular, the compliance with bus-protocols and other interactions with the HSM, the memory or other external components are potential candidates for formal verification. We will not focus on the HSM's cryptographic features and other non-functional aspects, as they are notoriously hard to formally define in the first place. Rather than prove weak and unsatisfying security properties we will concentrate on verifying that the addition of hardening measures and cryptographic extensions preserves correctness of the CPU and core components as sketched above.

[1] https://github.com/SymbioticEDA/riscv-formal/tree/master/cores/VexRiscv

7 Summary

This report contains an overview of use cases, requirements, development tools and means of verification for an open hardware security module. The components to be produced will address two objectives. The first is to make more open tools available to industry in order to support innovation and value creation. The second is to increase the security of open tools. International supply chains are susceptible to interference and attack by malicious actors. Also, hardware may be manipulated later, e.g. criminals may do a side-channel attack on an HSM used in automotive.

Open source development tools and free and permissive licensing can reduce the impact of trading disputes and drive competition as well as academic research, especially in the context of IT-security. For the software sector, open source tools and operating systems already enjoy wide-spread use by industry, academia and individuals alike. In the context of hardware development, this process has also started, e.g. with regard to RISC-V processors, but has only begun in the area of hardware development tools. Security is of vital importance to the industry, however, currently available open source tools are mostly concerned with functionality and ease of use, not with lack of vulnerabilities or formal verification. Most aspects of security and are therefore unsuited for many industrial applications.

This project aims to push towards this threshold by establishing and demonstrating an open source design flow for RISC-V security-aware hardware development. The design of a hardware security module (HSM) is an excellent demonstration for the security-aware hardware development tools to be established by this project. The use-cases described in Chapter 3 are common for applications in the automotive industry and likely share requirements with many other industrial applications. The ever-increasing connectivity leads to a more wide-spread use of cryptography to protect confidentiality and authenticity of data. This in turn leads to a requirement for resistance against the implementation specific attacks discussed in Section 3.3. In light of the ongoing back-and-forth between novel techniques for attack and defense, it is beneficial to provide the broader community with the tools required for participating in this process. For the actual demonstration of the design flow, an HSM will be implemented, following the TPM 2.0 specification. Together with an exemplary application processor, multiple instances of the HSM can be used to demonstrate security functionalities fulfilling the requirements specified in Chapter 4. The requirements specific to the HSM include secure boot, secure update, confidential data access, and secure access control.

The hardware component of the HSM demonstrator will be based on a RISC-V processor that implements the rv32i base instruction set, complemented by the needed

standard extensions and some suitable custom extensions. The design of such an HSM shows the broad variety of challenges for the corresponding tool-flow. In particular, physical access to securely stored data is a major threat, thus methods for hardening the hardware against such side-channel attacks (SCA) have to be provided. There are several hardware-based threads, such as timing or power SCAs that can be addressed by implementing dedicated protection designs into the functional circuitry. These implementations need to be implemented automatically by using open-source tools. In addition, formal methods must be applied to verify the implemented protection schemes and functionality aspects as well as showing the absence of undocumented hardware features such as Hardware Trojans. Hence, formal methods should be integrated into every step of the design process. Therefore, each of these steps cannot be solved independently and the chosen tools used to do so should be highly interoperable.

The choice of a Hardware Description Language depends on multiple factors: We need a language with a high abstraction level that still allows us to have full control over the generated hardware. This eliminates low-level HDLs such as VHDL and Verilog, but also the high level synthesis approach, and leaves the Hardware Construction Languages. The most promising candidates are Chisel and SpinalHDL as they are very well-made with an active community, and there exist open-source processors written in these two languages (Rocket and VexRiscv). The VexRiscv has allready proven itself to be highly configurable and efficient is therefore the implementation of choice. As a consequence SpinalHDL will be used and in collaboration with the developer some vital formal verification functionalities have already been added.

Regarding the tools for hardware development, our available choices are very limited, as we want to rely on open-source tools wherever possible. The Verilator project seems to be the most suited simulator as it is open-source, fast, and supports Verilog, which can be generated from SpinalHDL. Our choice of a synthesis tool is Yosys because it offers the best functionalities among the open-source synthesis tools and has support for ASIC and FPGA targets. Working well together with Yosys and being open-source, nextpnr is the ideal place&route tool targeting FPGAs for us. Open-source place&route tools targeting ASICs are currently being evaluated, on the basis of those tools included in the OpenROAD project[1]. The following processing steps are determined by the semiconductor fab in respect to the booked production technology, in our case it will be a 120nm process. Nevertheless, it will be necessary to verify the functionality of the physically produced chip and its implemented tamper protection mechanisms.

For software development, there exists a plethora of tools that are open source and that can be used to implement the software components that are required. The HSM firmware can be implemented in a low-level programming language such as C or Rust, and the existing open source tools can be used throughout the entire development process. This will most likely involve using the GNU Compiler Collection[2] which already supports the RISC-V ISA and its extension with custom instructions.

[1] https://github.com/The-OpenROAD-Project/OpenROAD
[2] https://github.com/riscv/riscv-gnu-toolchain

Alternatives however exist, the LLVM project[3] also has support for the RISC-V ISA and is the basis for several C and Rust compilers.

For verification, we plan to rely on existing technology such as bounded model check to prove the correctness of VexRisv with respect to the RISC-V ISA. In addition, we plan to verify the compliance with the bus protocol. For the hardening and cryptographic measures, we plan to verify their equivalence with the unextended core. All in all, our verification efforts will provide the developed solution with the necessary quality commensurate with the requirements of the automotive industry.

[3] https://github.com/llvm/llvm-project

Bibliography

[AES] Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce. Nov. 2001.

[AKS21] Mahya Morid Ahmadi, Faiq Khalid, and Muhammad Shafique. "Side-Channel Attacks on RISC-V Processors: Current Progress, Challenges, and Opportunities". In: CoRR abs/2106.08877 (2021). arXiv: 2106.08877. URL: https://arxiv.org/abs/2106.08877.

[Ami+18] Elham Amini et al. "Assessment of a Chip Backside Protection". In: Journal of Hardware and Systems Security 2 (Dec. 2018), pp. 345–352. DOI: 10.1007/s41635-018-0052-3.

[AW19] Krste Asanovic and Andrew Waterman. "The RISC-V Instruction Set Manual". In: Privileged Architecture, Document Version 20190608-Priv-MSU-Ratified. Vol. 2. RISC-V Foundation, 2019.

[Bar+15] Gilles Barthe et al. "Verified Proofs of Higher-Order Masking". In: Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015, pp. 457–485.

[Bar+16] Gilles Barthe et al. "Strong Non-Interference and Type-Directed Higher-Order Masking". In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. Ed. by Edgar R. Weippl et al. ACM, 2016, pp. 116–129.

[BT19] Swarup Bhunia and Mohammad H. Tehranipoor. Hardware security: a hands-on learning approach. Cambridge, MA: Morgan Kaufmann Publishers, 2019. ISBN: 978-0-12-812477-2.

[Cas+20] Gaëtan Cassiers et al. "Hardware Private Circuits: From Trivial Composition to Full Verification". In: IACR Cryptol. ePrint Arch. 2020 (2020), p. 185.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. "Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference". In: IEEE Trans. Inf. Forensics Secur. 15 (2020), pp. 2542–2555.

[CZ06] Zhimin Chen and Yujie Zhou. “Dual-Rail Random Switching Logic: A Countermeasure to Reduce Side Channel Leakage”. In: Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings. Ed. by Louis Goubin and Mitsuru Matsui. Vol. 4249. Lecture Notes in Computer Science. Springer, 2006, pp. 242–254.

[Dah21] Nitin Dahad. Intel Looking to Buy SiFive for $2bn. 2021. URL: https://www.eetimes.com/intel-looking-to-buy-sifive-for-2bn/ (visited on 08/09/2021).

[DGH19] Elke De Mulder, Samatha Gummalla, and Michael Hutter. “Protecting RISC-V against Side-Channel Attacks”. In: Proceedings of the 56th Annual Design Automation Conference 2019. DAC ’19. Las Vegas, NV, USA: Association for Computing Machinery, 2019. ISBN: 9781450367257. DOI: 10.1145/3316781.3323485. URL: https://doi.org/10.1145/3316781.3323485.

[ECC] Don Johnson, Alfred Menezes, and Scott Vanstone. “The elliptic curve digital signature algorithm (ECDSA)”. In: International journal of information security 1.1 (2001), pp. 36–63.

[Fau+18] Sebastian Faust et al. “Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model”. In: IACR Trans. Cryptogr. Hardw. Embed. Syst. 2018.3 (2018), pp. 89–120.

[GIB18] Hannes Groß, Rinat Iusupov, and Roderick Bloem. “Generic Low-Latency Masking in Hardware”. In: IACR Trans. Cryptogr. Hardw. Embed. Syst. 2018.2 (2018), pp. 1–21.

[GM18] Hannes Groß and Stefan Mangard. “A unified masking approach”. In: J. Cryptogr. Eng. 8.2 (2018), pp. 109–124.

[GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. “An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order”. In: Topics in Cryptology - CT-RSA 2017 - The Cryptographers’ Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings. Ed. by Helena Handschuh. Vol. 10159. Lecture Notes in Computer Science. Springer, 2017, pp. 95–112.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. “Electromagnetic Analysis: Concrete Results”. In: Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings. Ed. by Çetin Kaya Koç, David Naccache, and Christof Paar. Vol. 2162. Lecture Notes in Computer Science Generators. Springer, 2001, pp. 251–261.

[GST17] Daniel Genkin, Adi Shamir, and Eran Tromer. “Acoustic Cryptanalysis”. In: J. Cryptol. 30.2 (2017), pp. 392–443.

[Hru18] Joel Hruska. ARM Kills Its RISC-V FUD Website After Staff Revolt. 2018. URL: https://www.extremetech.com/computing/273236-arm-kills-its-risc-v-fud-website-after-staff-revolt (visited on 08/10/2021).

[HS13] Michael Hutter and Jörn-Marc Schmidt. "The Temperature Side Channel and Heating Fault Attacks". In: Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers. Ed. by Aurélien Francillon and Pankaj Rohatgi. Vol. 8419. Lecture Notes in Computer Science. Springer, 2013, pp. 219–235.

[ISO21] ISO/SAE FDIS 21434:2021(E). Road vehicles — Cybersecurity engineering. Standard. Geneva, CH: International Organization for Standardization, Aug. 2021.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. "Private Circuits: Securing Hardware against Probing Attacks". In: Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Springer, 2003, pp. 463–481.

[Jay09] Ramkumar Jayaseelan. "Application-specific Thermal Management of Computer Systems". PhD thesis. National University of Singapore, 2009.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. "Differential Power Analysis". In: Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 388–397.

[KK10] I. Koren and C.M. Krishna. Fault-Tolerant Systems. Elsevier Science, 2010. ISBN: 9780080492681. URL: https://books.google.de/books?id=o%5C_Pjbo4Wvp8C.

[Koc96] Paul C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings. Ed. by Neal Koblitz. Vol. 1109. Lecture Notes in Computer Science. Springer, 1996, pp. 104–113.

[Lan13] Ralph Langner. To kill a centrifuge. Tech. rep. The Langner Group, 2013. URL: https://www.langner.com/wp-content/uploads/2017/03/to-kill-a-centrifuge.pdf.

[Mar+21] Ben Marshall et al. "The design of scalar AES Instruction Set Extensions for RISC-V". en. In: IACR Transactions on Cryptographic Hardware and Embedded Systems (2021), pp. 109–136. ISSN: 2569-2925. DOI: 10.46586/tches.v2021.i1.109-136. URL: https://tches.iacr.org/index.php/TCHES/article/view/8729 (visited on 07/15/2021).

[Mor11] Shailesh More. “Aging Degradation and Countermeasures in Deep-submicrometer Analog and Mixed Signal Integrated Circuits”. PhD thesis. Lehrstuhl für Technische Elektronik der Technischen Universität München, 2011.

[MS06] Stefan Mangard and Kai Schramm. “Pinpointing the Side-Channel Leakage of Masked AES Hardware Implementations”. In: Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings. Ed. by Louis Goubin and Mitsuru Matsui. Vol. 4249. Lecture Notes in Computer Science. Springer, 2006, pp. 76–90.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. “Threshold Implementations Against Side-Channel Attacks and Glitches”. In: Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings. Ed. by Peng Ning, Sihan Qing, and Ninghui Li. Vol. 4307. Lecture Notes in Computer Science. Springer, 2006, pp. 529–545.

[OFl20] Colin O’Flynn. BAM BAM!! On Reliability of EMFI for in-situ Automotive ECU Attacks*. Tech. rep. Dalhousie University, Halifax, Canada, 2020. URL: https://eprint.iacr.org/2020/937.pdf.

[PM05] Thomas Popp and Stefan Mangard. “Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints”. In: Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings. Ed. by Josyula R. Rao and Berk Sunar. Vol. 3659. Lecture Notes in Computer Science. Springer, 2005, pp. 172–186.

[Rep+15] Oscar Reparaz et al. “Consolidating Masking Schemes”. In: Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I. Ed. by Rosario Gennaro and Matthew Robshaw. Vol. 9215. Lecture Notes in Computer Science. Springer, 2015, pp. 764–783.

[RSA] PKCS #1: RSA Cryptography Standard. RSA Data Security, Inc. Version 2.0. Sept. 1998.

[SHA93] Secure Hash Standard. National Institute of Standards and Technology, NIST FIPS PUB 180, U.S. Department of Commerce. May 1993.

[Sri+04] Jayanth Srinivasan et al. “The Impact of Technology Scaling on Lifetime Reliability”. In: Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN ’04) (2004), pp. 177–186.

[SS06] Daisuke Suzuki and Minoru Saeki. “Security Evaluation of DPA Countermeasures Using Dual-Rail Pre-charge Logic Style”. In: Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings. Ed. by Louis Goubin and Mitsuru Matsui. Vol. 4249. Lecture Notes in Computer Science. Springer, 2006, pp. 255–269.

[SVZ19] Ben Seri, Gregory Vishnepolsky, and Dor Zusman. Bleedingbit: The hidden attack surface within BLE chips. Tech. rep. Armis Inc., 2019. URL: https://info.armis.com/rs/645-PDC-047/images/Armis-BLEEDINGBIT-Technical-White-Paper-WP.pdf.

[SX20] Frans Sijstermans and Joe Xie. Keynote: NVIDIA's secure RISC-V processor. 2020. URL: https://youtu.be/7Lx3692cbAg (visited on 08/09/2021).

[Tir+05] Kris Tiri et al. "Prototype IC with WDDL and Differential Routing - DPA Resistance Assessment". In: Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings. Ed. by Josyula R. Rao and Berk Sunar. Vol. 3659. Lecture Notes in Computer Science. Springer, 2005, pp. 354–365.

[TIS] Trusted Computing Group. "TCG PC Client Specific TPM Interface Specification (TIS)". In: Specification Version 1.3 1 (2013). https://trustedcomputinggroup.org/resource/pc-client-platform-tpm-profile-ptp-specification/.

[TSN17] Dharmesh Thakker, Max Schireson, and Dan Nguyen-Huu. Tracking the explosive growth of open-source software. 2017. URL: https://techcrunch.com/2017/04/07/tracking-the-explosive-growth-of-open-source-software/ (visited on 08/09/2021).

[TV04] Kris Tiri and Ingrid Verbauwhede. "A Dynamic and Differential CMOS Logic Style to Resist Power and Timing Attacks on Security IC's". In: IACR Cryptol. ePrint Arch. 2004 (2004), p. 66.

[TV05] Kris Tiri and Ingrid Verbauwhede. "A VLSI Design Flow for Secure Side-Channel Attack Resistant ICs". In: 2005 Design, Automation and Test in Europe Conference and Exposition (DATE 2005), 7-11 March 2005, Munich, Germany. IEEE Computer Society, 2005, pp. 58–63.

[Ustr21] Upstream Security's 2021 Global Automotive Cybersecurity Report. https://upstream.auto/2021report/. Detroit, 2021.

[Vis+00] Ram Viswanath et al. "Thermal Performance Challenges from Silicon to Systems". In: Intel Technology Journal 4.3 (2000), pp. 1–16.

[WB08] Mark White and Joseph B. Bernstein. Microelectronics Reliability: Physics-of-Failure Based Modeling and Lifetime Evaluation. Tech. rep. Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA 91109: National Aeronautics and Space Administration, 2008. URL: https://nepp.nasa.gov/files/16365/08_102_4_%20JPL_White.pdf.