FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Embedded Security Analysis with Emphasis on Critical Infrastructures

DISSERTATION

zur Erlangung des akademischen Grades

Doktor/in der technischen Wissenschaften

eingereicht von

Dipl. Ing. Markus Kammerstetter, BSc.
Matrikelnummer 0226196

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof.Dr. Wolfgang Kastner

Diese Dissertation haben begutachtet:

(Ao. Univ. Prof. Dr. Wolfgang Kastner)	(Prof. Dr.-Ing. Tim Güneysu)

Wien, 26.07.2016

(Dipl. Ing. Markus Kammerstetter, BSc.)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Embedded Security Analysis with Emphasis on Critical Infrastructures

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor/in der technischen Wissenschaften

by

Dipl. Ing. Markus Kammerstetter, BSc.
Registration Number 0226196

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof.Dr. Wolfgang Kastner

The dissertation has been reviewed by:

(Ao. Univ. Prof. Dr. Wolfgang Kastner)	(Prof. Dr.-Ing. Tim Güneysu)

Wien, 26.07.2016

(Dipl. Ing. Markus Kammerstetter, BSc.)

Erklärung zur Verfassung der Arbeit

Dipl. Ing. Markus Kammerstetter, BSc.
Rienoesslgasse 14/17, 1040 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Wien, am 26.07.2016) (Unterschrift Verfasser)

Acknowledgements

I am deeply grateful to my advisor Wolfgang Kastner for making this thesis and the involved work behind it possible. I would like to thank the entire Automation Systems Group for providing me with the freedom and support in establishing and maintaining a Hardware Security Lab including many of its challenges such as the need for more room or the inherent challenges when trying to set up a 1 ton instrument in an old building with limited ceiling load capacity. In that regard, my thanks go especially to Ruth Fochtner and Wolfgang Kastner who have always been there for me over the last several years. A great acknowledgement is dedicated to my former Secure Systems Lab colleagues including Christopher Kruegel, Engin Kirda, Gilbert Wondracek, Christian Platzer, Thorsten Holz, Clemens Kolbitsch, Matthias Neugschwandtner, Paolo Milani Comparetti, Martina Lindorfer and Adrian Dabrowski. I want to show my appreciation to my colleagues at SBA Research including Edgar Weippl, Martin Schmiedecker, Georg Merzdovnik and Markus Huber for the ongoing collaboration and teaching activities. I had the pleasure to work together with other researchers including many of the AIT Safety and Security Department such as Lucie Langer, Florian Skopik, Paul Smith, Friederich Kupzog and Thomas Bleier. Several bachelor- and master students supported and helped me during my experiments and the necessary implementations. I would like to express my gratitude to my current hardware security team including Markus Muellner, Daniel Burian, Stefan Riegler, Viktor Ullmann and Christian Kudera. I would further like to thank the longtime customers of my security consulting company Trustworks KG for indirectly providing we with funding for many of my research activities. Special thanks go to my friends and my family including my grand parents Berta and Herbert as well as my uncle Heribert who has been an inspiration to me from childhood on. I would like to express my deepest gratitude to Gerda for supporting me, tolerating my sometimes odd research activities at home including having a lab fume hood in the living room and the long working hours I have been in the lab instead of spending the time with her. I would like to express my appreciation to Markus Kuhn, Sergei Skorobogatov and Christopher Tarnovsky for getting me started in the field of IC reverse engineering in the first place. Lastly, I would like to thank the people I forgot to mention, the discovery of $C_8H_{10}N_4O_2$, HNO_3, H_2SO_4, C_3H_6O and the invention of nanotechnology, Integrated Circuits (ICs), the Scanning Electron Microscope (SEM) and the Focused Ion Beam (FIB).

Abstract

Embedded systems are ubiquitously used today. From everyday electronic consumer products to critical domains such as medical devices, *Electronic Control Units (ECUs)* in cars or critical infrastructure field components, their possible fields of application are manifold. While some of these systems are highly security critical and successful attacks could lead to disastrous effects, in the past their exposure to potential attackers was limited due to a lack of widely accessible communication interfaces. For instance, the ECUs in cars used to be interconnected but there were no wireless uplink connections to connect the car to the Internet, to the user's smart phone or to other cars via vehicle-to-vehicle communication. Many of today's embedded systems were thus designed with a focus on functionality and safety. Security was not a major concern. However, today there is an ongoing paradigm shift from considered *dumb* devices to highly interconnected *smart* devices. With *Industry 4.0*, there is another ongoing industrial revolution that transforms traditional production systems to ICT enabled *smart factories*. On the *Internet of Things (IoT)*, an increasing number of everyday embedded devices gets connected to the Internet. Cars often include multiple Internet uplink connections, medical devices such as pacemakers can be adjusted over wireless interfaces and networked devices within the consumer's premises and the power grid pave the way for green energy consumption and the *smart grid*. Considering that many potentially insecure systems now become internetworked and accessible to potential attackers, this leaves many critical systems such as smart critical infrastructures at stake.

While a secure system can only be achieved if security is of major concern from the very beginning and a secure life-cycle involving processes such as secure design, secure implementation and continuous security assessments is followed, recent publications have shown that embedded systems do not cope well with this security demand. Ultimately, not only the system manufacturers but also the system owners and operators need ways to assess, manage and test the security of networked embedded systems. This thesis focuses on embedded system security within the smart grid critical infrastructure.

The problem of smart grid security assessment and management is addressed by presenting an architecture driven approach allowing operators such as utilities to identify high-risk smart grid components in their grid instances, to select those components for detailed technical security audits and to subsequently mitigate security threats. Testing the security of high-risk embedded smart grid field components is still considered to be challenging and more time consuming in comparison to off-the-shelf PC based systems. A major cause is that prevalent vulnerability discovery techniques on embedded systems are still largely based on static analysis.

To address some of these shortcomings, the use of emulators with proprietary peripheral device communication forwarding is investigated to enable dynamic security analysis approaches such as

fuzz testing. The thesis introduces PROSPECT, a proxy capable of tunneling arbitrary peripheral hardware accesses from within a virtual analysis environment to the embedded system under test. Our system thus enables the analysts to leverage any powerful dynamic analysis techniques of their choice to discover vulnerabilities on embedded devices with minimal effort. In addition, the use of firmware program state approximation is explored to allow caching device responses within the PROSPECT system. Our case study shows that during security testing, future implementations of peripheral device caching could pave the way for powerful functions such as snapshotting, test parallelization or testing without physical access to the embedded system.

Since security testing of embedded firmware is only feasible if the firmware can be extracted from the embedded device in the first place, physical attacks are described that can be applied to embedded smart grid devices. From these physical attacks, the use of limited Integrated Circuit (IC) reverse engineering techniques is explored to discover proprietary test modes in silicon. Once the test mode is known to the analysts, it is often possible to extract the firmware from the device and subsequently perform firmware security tests.

Finally, besides embedded firmware extraction and analysis, future smart grid protocols will involve cryptographic authentication mechanisms that need to be tested and practically evaluated as well. Since no established cryptographic smart grid authentication protocols exist yet, the thesis presents a highly efficient FPGA cluster architecture and implementation of a brute-force attack on the well known WPA2-Personal authentication protocol instead. Our results indicate that a very high attack performance can be achieved and our approach would be suitable to test the practical security of future smart grid authentication protocols.

The work presented in this thesis thus provides a holistic embedded security analysis approach for critical smart grid components ranging from architecture modeling, risk assessment and security management over firmware extraction and firmware security analysis to the practical analysis of cryptographic authentication protocols.

Kurzfassung

Eingebettete Systeme werden heute allgegenwärtig eingesetzt. Ihre vielfältigen Einsatzgebiete reichen von tagtäglicher Unterhaltungs- und Haushaltselektronik, bis hin zu kritischen Anwendungsbereichen, wie medizinischen Geräten, Steuergeräten in Fahrzeugen oder den Feldkomponenten von kritischen Infrastrukturen. Besonders bei sicherheitskritischen Geräten können jedoch erfolgreiche Angriffe desaströse Auswirkungen mit sich bringen. In der Vergangenheit waren derartige Geräte oft kaum vernetzt, sodass deren Exponiertheit gegenüber Angreifern aufgrund der fehlenden physischen Zugriffsmöglichkeit beschränkt war. So waren etwa die Steuergeräte in Fahrzeugen auch schon in der Vergangenheit miteinander verbunden, jedoch gab es keine externe Schnittstellen, die das System etwa zum Internet, zum Smartphone oder drahtlos zu anderen Fahrzeugen vernetzten. Viele der heute im Einsatz befindlichen eingebetteten Systeme wurden somit mit einem Fokus auf Funktionalität und Betriebssicherheit entwickelt und deren Sicherheit gegenüber böswilligen Angriffen wurde nur nebensächlich oder gar nicht berücksichtigt. Aktuell ist jedoch ein Paradigmenwechsel im Gange, sodass Geräte zunehmend vernetzt und dadurch intelligenter werden. Man spricht von sogenannten *Smart Devices*. Im Bereich *Industrie 4.0* findet aktuell eine weitere industrielle Revolution statt, die traditionelle Produktions-Systeme zu IKT gestützten *Smart Factories* transformiert. Im *Internet der Dinge* werden zunehmend tagtägliche Geräte mit dem Internet verbunden. Viele Fahrzeuge haben bereits mehrfache Verbindungen zum Internet, medizinische Geräte, wie Herzschrittmacher, lassen sich drahtlos parametrieren und intelligente Geräte im Haushalt und im Stromnetz ebnen den Weg zum grünen Energieverbrauch. Auf der Kehrseite heißt dies jedoch auch, dass viele kritische sowie potenziell unsichere Systeme, wie etwa im Bereich der Smart Grids, nun stark miteinander vernetzt und damit leichter von außen angreifbar werden.

Ein sicheres Gesamtsystem kann nur dann erreicht werden, wenn die Sicherheit des Systems von Beginn an ein wichtiger Gesichtspunkt war und ein sicherer Entwicklungs- und Lebenszyklus eingehalten wird. Dieser beinhaltet etwa ein sicheres Systemdesign, eine sichere Implementierung dieses Designs und regelmäßige Sicherheitsüberprüfungen (Audits) der im Einsatz befindlichen Systeme. Aktuelle Publikationen haben jedoch gezeigt, dass eingebettete Systeme diesen Sicherheitsanforderungen nur schlecht nachkommen. Sowohl die Hersteller wie auch die Systembesitzer und Betreiber benötigen Möglichkeiten, um vernetzte eingebettete Systeme nicht nur auf Sicherheit hin zu testen, sondern deren Sicherheitsniveau im Einsatz auch hoch halten zu können. Der Fokus dieser Arbeit liegt auf Sicherheitsuntersuchungen von eingebetteten Systemen im Umfeld der intelligenten Stromnetzinfrastrukturen. Die Problematik der Risikobewertung und des Risikomanagements wird mittels eines architekturgestützten Ansatzes behandelt, der es etwa Netzbetreibern ermöglicht, besonders risikobehaftete Komponenten in intelligenten Stromnetzen

zu identifizieren, für weitergehende technische Sicherheitsaudits auszuwählen und folglich deren Sicherheitsrisiken zu senken. Im Vergleich zu handelsüblichen Computersystemen gilt die Durchführung von technischen Sicherheitsaudits auf eingebetteten Systemen heute noch als äußerst anspruchsvoll und zeitaufwändig. Einer der Hauptgründe ist die Schwierigkeit dynamische Analyseverfahren einzusetzen, sodass Sicherheitsanalysen auf diesen Systemen noch weitestgehend auf statischen Analyseverfahren aufbauen.
Die vorliegende Arbeit behandelt diese Herausforderung, indem der Einsatz von Emulationstechniken in Zusammenhang mit der Weiterleitung der Kommunikation zu peripheren Hardwarebausteinen untersucht wird. Durch unseren Ansatz werden etwa bestehende dynamische Analyseverfahren, wie *Fuzz Testing*, maßgeblich erleichtert. Die vorliegende Arbeit stellt PROSPECT vor, ein System, das als transparenter Proxy für beliebige Peripheriekommunikation aus einer virtuellen Analyseumgebung heraus zu dem im Test befindlichen eingebetteten System agiert. PROSPECT ermöglicht es, auf eingebetteten Systemen Sicherheitsanalysen mit starken dynamischen Analyseverfahren durchzuführen und mit geringem Aufwand Schwachstellen zu identifizieren. Zusätzlich wird der Einsatz eines Cache Speichers in Kombination mit einer Approximierung des Programmzustands betrachtet, um die Peripheriekommunikation zwischenzuspeichern. Die Fallstudie zeigt, dass die vorgestellte Technik einen zukünftigen Weg in Richtung sehr mächtiger Verfahren wie Snapshotting, Test-Parallelisierung oder dem Testen ohne physischen Zugriff auf das eingebettete System ermöglichen könnte.
Um vorgestellte Sicherheitsanalyse-Techniken überhaupt erst auf der Firmware von eingebetteten Systemen nutzen zu können, ist es ebenso notwendig, die Firmware aus den Geräten zu extrahieren. Zu diesem Zweck wird den Einsatz von physischen Hardware-Angriffen auf die eingebetteten Systeme in intelligenten Stromnetzen behandelt. Insbesondere wird der Einsatz von Mikrochip Reverse Engineering Techniken betrachtet, um versteckte Testmodi im Chip auffinden zu können. Identifizierte Testmodi können im weiteren oft dazu verwendet werden, um die Firmware aus dem Chip zu extrahieren und folglich auf ihre Sicherheit hin zu analysieren.
Neben der Firmware Extraktion und deren Sicherheitsanalyse wird es in zukünftigen intelligenten Stromnetzen auch erforderlich sein, kryptografische Authentifikationsmechanismen auf ihre praktische Sicherheit hin überprüfen zu können. Wenngleich aktuell keine weit verbreiteten entsprechenden Verfahren für intelligente Stromnetze existieren, wird in dieser Arbeit der Einsatz einer hoch-effizienten FPGA Architektur und deren Implementierung erprobt, um am Beispiel des weit verbreiteten kryptografischen WPA2-Personal Authentifikationsverfahrens entsprechende praktische Sicherheitstests durchführen zu können. Die Ergebnisse zeigen, dass sich mit dem System hohe Angriffsgeschwindigkeiten realisieren lassen, die sich etwa auch für praktische Sicherheitstests von zukünftigen Smart Grid Authentifikationsverfahren eignen würden.
Die in dieser Dissertation vorgestellten Arbeiten beschreiben somit einen gesamtheitlichen Ansatz zur Sicherheitsanalyse von eingebetteten Systeme in intelligenten Stromnetzen und reichen von der Architektur-Modellierung, Risikobewertung und dem Security Management über Firmware Extraktion und Firmware Sicherheitsanalyse, bis hin zur praktischen Analyse von eingesetzten kryptografischen Authentifikationsprotokollen.

Contents

CHAPTER 1

Introduction

"We need to be able to verify the software that controls our lives"

Bruce Schneier on 'Volkswagen and Cheating Software'

1.1 Problem Statement

Embedded systems are an integral part of almost every electronic product today. From consumer electronics, the *Internet of Things (IoT)* and ICT enabled production systems (i.e., *Industrial Internet*) over automotive systems such as Electronic Control Units (ECUs) to payment systems and critical infrastructure devices, their possible fields of application are manifold. Embedded systems in their different forms and sizes are so widespread that hundreds of those devices can be found in our living environments today [4]. While especially in industrial production systems, medical appliances and critical infrastructures the security requirements are high, recent publications have shown that embedded systems do not cope well with this demand [58, 65, 88, 119]. One of the key reasons is that embedded systems are being less scrutinized as embedded security analysis is considered to be much more time consuming and challenging in comparison to off-the-shelf PC based systems. At the same time, an ongoing trend towards interconnected embedded systems can be observed. While just a few years ago your coffee machine, car, electricity meter and boiler at home were all considered to be *dumb* devices with embedded systems in them, today many of those devices get *smart* by interconnecting them. The coffee machine gets just another Internet of Things (IoT) device that can now be controlled via an Internet cloud service from anywhere in the world with a smart phone. A car has multiple uplink connections for the media center, the navigation system, to locate and control some of its devices with your smart phone or to call for help in case of an accident. The electricity meter and boiler become smart devices by exchanging information with each other and renewable energy sources such as the photovoltaic (PV) modules on the roof of your house. This paves the way for ecologically *green* energy consumption, for instance, by scheduling the water heating in the

boiler in a way that solar energy or reserves at the energy supplier are used more efficiently. If you have a weak heart, chances are good that you already have a state-of-the-art pacemaker that supports wireless communication. Over the wireless link it can be connected to the Internet so that a medical cloud service can monitor your heart condition and automatically alarm your physician in case of an emergency [41]. However, from a security perspective, it means that a high number of embedded systems that have never been designed with security in mind can now be accessed remotely by attackers over a network. Considering the broad attack surface of the embedded systems in cars [20], it has only recently been demonstrated how attackers can take remote control of an unaltered driving car over the Internet by exploiting a flaw in the car's media center [10, 73, 76]. The wireless interface in typical pacemakers is not well protected either and attackers can leverage replay attacks with a low-cost Software Defined Radio (SDRs) to deliver life threatening electric shocks to the heart muscle [41]. Similarly, an attacker being able to remotely turn off the power supply of an entire house via an insecure smart meter might be considered as a tremendous nuisance to the home owner at first. However, if the attacker manages to switch off a large number of homes within a city at the same time (i.e., a *mass attack*) the security impact becomes disastrous. The sudden large scale energy consumption drop would result in a rise of the power grid voltage beyond safe limits. To protect the power grid from even further damages, the overvoltage would trigger emergency shutdown switches within network segments of the power grid. However, since these emergency shutdowns would cause an even larger energy consumption drop it is likely that a cascade effect is caused leading to a full scale blackout. The longer the blackout remains (and possibly even prolonged by repeated attacks on the embedded infrastructure), the more life threatening the situation becomes as many of our other critical infrastructures such as the water or gas supplies depend on electrical power and their related actors. Typical scenarios that often lead to insecure networked embedded devices within critical systems are described in the following:

System Manufacturer

The system manufacturer is well established in its domain and the currently rolled out *smart* products (i.e., interconnected embedded systems) are only a small part of their entire product range. The product was developed over a long time period and is largely based on soft- and hardware components that can be found in the manufacturer's other products as well to decrease the time-to-market (*code reuse*). Even though the manufacturer has extensive experience in his domain, the product development focus lies primarily on functionality and safety. Adding ICT technologies to the products is not entirely new to the system manufacturer, but so far those communication interfaces have never been exposed to the general public and potential adversaries. As a result, most products have no intrinsic security design, the use of insecure proprietary communication protocols is common, the developers have only marginal experience in secure design and security has not been a major concern for the manufacturer so far. Since secure design, development and security in general is a new field to the manufacturer, it is likely that the new *smart* product range is relatively easy to exploit by attackers. At the same time, it is common that there is no budget for security testing as those security tests are considered to be unnecessary expenses that do not directly generate revenue for the manufacturer.

End-users and System Operators

Depending on the smart product type, the potential insecure networked embedded systems are either used directly or indirectly by end-users or system operators. While a car with all its smart devices is directly used by a person, a pacemaker is merely indirectly used by the patient as even though it is physically located in the patient's body, it is still the doctor who communicates and interacts with the device. In contrast, critical infrastructures are operated by system operators (i.e., the Distribution System Operator (DSO) in the case of the power grid) and even though some of the infrastructure's networked embedded devices such as smart meters may be located within their premises, the end-users typically neither own nor can they communicate with them. In general, especially for critical systems, users will have a strong interest that the devices are secure. However, they might not know which components are especially exposed to security risks and the financial means to conduct independent in-depth security tests could be low or even non existent. A private car owner will thus need to trust the manufacturer that the embedded devices are secure but if the car manufacturer failed to secure them, they will remain insecure and prone to attacks. Even if a system owner such as a DSO has the financial means to conduct independent security audits and tests, the extent of those tests will typically be strongly limited by costs resulting in time and effort constraints for the security analyst.

Security Analyst

The security analyst needs to overcome many challenges to conduct an in-depth technical security audit on a networked embedded system. The aim of the analysis is to gain deep technical insights into the implementation of the security critical functions so that the resulting security audit report is based on a well founded accurate technical analysis. The analyst is under the pressure to deliver the security analysis within the limited time and effort that was indirectly agreed upon with the client through the cost of the security audit. However, in contrast to the security tests of PC based software, embedded security tests bring many additional challenges the analyst needs to overcome first. For instance, the firmware might not be available to the analyst in the first place since it may be locked into a readout protected chip inside the embedded system. If the analyst is able to extract the firmware, it is often infeasible to utilize powerful state-of-the-art dynamic analysis techniques ranging from fuzz testing to advanced dynamic taint analysis and symbolic or concolic execution [17, 94]. Instead, prevalent vulnerability discovery techniques on embedded systems are still largely based on static analysis [13, 59, 118]. Considering the typical time constraints and the many hurdles that need to be overcome, it is thus likely that the security analyst will spend most of the analysis effort not analyzing the actual security critical implementation but rather working on ways to overcome the prerequisites for the security tests. Given that the time and effort requirements for embedded security tests are much higher in comparison to security tests on PC based systems and the necessary analysis time is often not available due to its costs, the outcome of conducted embedded security tests may not be based on a well founded technical analysis. As a result, major vulnerabilities might not be discovered at all. Although attackers would need to overcome similar hurdles to discover exploitable vulnerabilities, in contrast to security analysts the difference is that they are not limited in analysis time.

In this work, we focus on security analysis of networked embedded systems within the critical *smart grid* infrastructure with the aim to overcome some of these shortcomings. Smart grids adhere to a hierarchical model. At the top, there are typically large scale classical IT systems and SCADA systems employing standard IT components. As a result, standard security management and assessment procedures can be applied to those systems. However, towards the field level, the smart grid heavily relies on embedded systems where the traditional security management processes described above are not applicable due do their highly specific nature and functionalities (Figure 1.1).

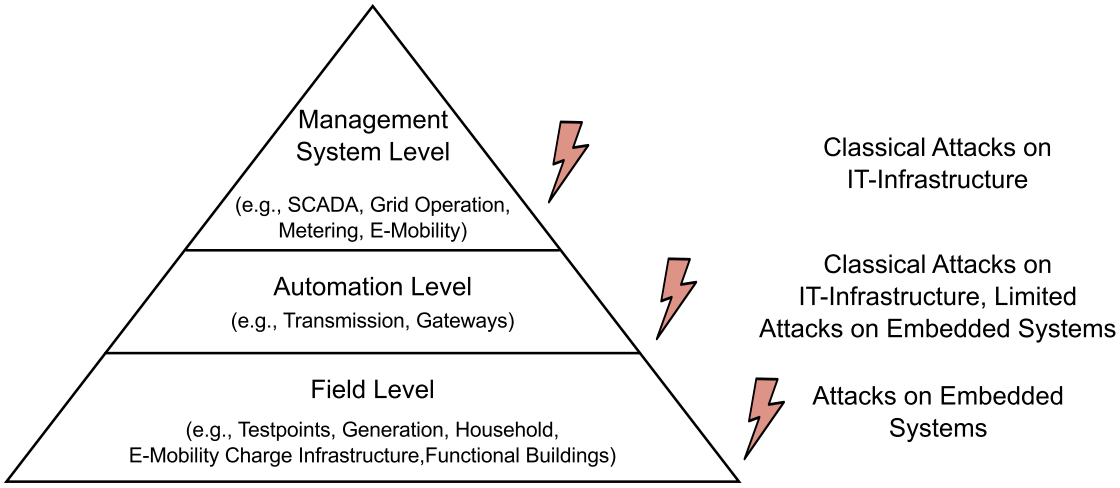

Figure 1.1: Attacks on the Smart Grid Hierarchy

To increase the overall security of those critical infrastructure embedded systems as well, a security assessment approach needs to be developed that follows respected security management procedures, but focuses on the (technical) architectures of these embedded systems-of-systems just the same. This approach paves the way to identify systems with high attack potentials that can be analyzed for security vulnerabilities. To conduct these embedded security audits, state-of-the-art procedures need to be extended and new techniques need to be explored in order to minimize the gap between the technical analysis techniques available for embedded systems and the existing ones available for off-the-shelf PC systems. Some of the major problem fields that need to be addressed are dynamic analysis approaches on embedded systems as well as the analysis of systems that adhere to a security-by-obscurity model, for instance, by hiding security critical information in silicon to avoid independent security analysis. This includes the common problem that the firmware is locked into a silicon chip and can not be read out by analysts to conduct independent in-depth security tests.

1.2 Aim of this Work

The aim of this thesis is to enable a holistic embedded system security analysis with special emphasis on today's most critical infrastructure: the electrical power grid that currently is under continuous transformation to the smart grid. Specifically, we aim to address persistent problems in the *architecture driven smart grid risk assessment*, *firmware extraction*, *dynamic firmware analysis* and *high-speed cryptographic attack* domains. The expected result of this thesis is a solid and practically usable architecture driven smart grid security assessment approach with a current and near future time perspective to reflect the smart grid systems in the field right now or systems that will be in the field in near future. Based on this approach, critical devices can be identified that have a high-risk potential for attacks. At this point, the embedded security analysis techniques developed within this thesis can be applied to those smart grid components in order to assess their technical risks and to identify potential security vulnerabilities that would be a threat to national critical infrastructure security. If the firmware can not be obtained from the device through conventional means (i.e., over programming or debug interfaces), we provide an outlook on how on-chip test modes can be utilized to extract the firmware.

Architecture Driven Smart Grid Risk Assessment and Security Management

In the first step, starting at classical security management processes, an architecture driven smart grid security assessment approach is developed that can be practically used by utilities. In contrast to existing methods, this approach will focus on the architectures and technologies in smart grids that are already in use or that will be used in near future. Based on this smart grid model, a practical architecture driven risk assessment approach is developed, allowing utilities to identify high-risk smart grid components that need to undergo further security analysis.

Firmware Extraction

On embedded devices, a trend in the "security-by-obscurity" direction can be observed. Security critical functions as well as the overall firmware can be either locked or hidden in the silicon implementation of integrated circuits such as microcontrollers. To enable embedded security analysis on these devices, in this thesis we explore the use of physical attacks to leverage test modes hidden on the chip for firmware extraction and subsequent security analysis.

Dynamic Firmware Analysis

While for software components running on off-the-shelf PC systems there is a wide range of existing security analysis and vulnerability discovery techniques, typical field components in the smart grid hierarchy are embedded components that cannot be analyzed with these tools [13,59,118]. Ultimately, independent deep-level embedded software security analysis can only be done if the firmware is available to the analyst. While for source code audits existing approaches are available, they also require the system manufacturer to disclose all of the implementation's firmware to the analyst. Unfortunately, in the typical case, the source code is not available to analysts and hence this thesis will focus on binary embedded code analysis. As one of

the prevalent shortcomings of embedded code analysis is the difficulty of dynamic analysis techniques on proprietary embedded firmware, in this thesis we explore emulation technologies and hardware communication forwarding methods to enable dynamic firmware analysis for vulnerability discovery.

High-Speed Cryptographic Attacks

Besides vulnerabilities in the embedded firmware itself, many network protocols are secured with cryptographic authentication protocols so that only authorized users can interact with the embedded device. Considering embedded system security with respect to outside attackers without knowledge of authentication credentials, the attack surface on the firmware is thus limited to the code fragments that implement the network message handling prior to and including the authentication routines. While authentication significantly improves overall system security, authentication protocols including cryptographic protocols are often either not properly implemented or the utilized credentials such as passwords are too weak. An attacker could thus leverage brute-force or time/memory tradeoff attacks to obtain system credentials within a reasonable time frame. In this work, we aim to explore FPGA cluster supported high-speed attacks on cryptographic systems to analyze the practical security of systems in the field.

1.3 State-of-the-Art and Related Work

Smart grids and smart grid technologies have been an ongoing research topic in the recent years. While these technologies can significantly boost the efficiency and use of green energy, the strong integration of ICT technologies raises the concern of potential security vulnerabilities and attacks. Various works discuss the structure, building blocks, applications and potential impact of smart grids [3, 111]. Others provide an overview of the currently developed technical standards [25]. Even though the European Union aims for the transformation of traditional power grids into smart grids within the near future, major security and privacy aspects still have not been sufficiently addressed [60, 98]. For instance, both the U.S. NIST and the European ENISA have released numerous guidelines on how to secure smart grid architectures [33, 81]. The Smart Grid Coordination Group formed by the European standards organizations CEN, CENELEC and ETSI has provided a comprehensive framework on smart grids in response to the EU Smart Grid Mandate M/490 [102]. Within the "Smart Grid Information Security (SGIS)" report five SGIS *Security Levels* to assess the criticality of smart grid components have been defined even though the assessment is carried out under the unrealistic assumption that no security controls whatsoever are in place. For smart metering, the German Federal Office for Information Security (BSI) has come up with Common Criteria Protection Profiles [15, 16]. While these documents are an important step in the right directions, they do not offer a holistic approach. NIST, for instance, only focuses on technologies and regulatory security requirements for U.S. smart grids. In contrast, the ENISA security measures focus on European smart grids, but just like NIST the guidelines are mostly high-level only and the actual technical smart grid implementation is not considered. Similarly, the BSI protection profiles do not provide a holistic approach either and the focus of these profiles lies on smart metering only which is just one of the building blocks

of a smart grid. Even worse, their Target of Evaluation (TOE) focuses on a very specific smart metering implementation that neither reflects the currently deployed smart metering systems nor is it legally binding for European countries other than Germany. In contrast to the existing approaches, we leverage smart grid modeling to obtain a holistic model of the system components in European smart grids with a current and near future perspective. Subsequent risk assessment is thus no longer on a high level but it rather focuses on the actual smart grid implementation. Regarding risk assessment, Lu et al. present persistent security threats in the smart grid [72], while Ray et al. provide a more formal approach to smart grid risk management in general [92]. The differences between the risk modeling of traditional power grids and the smart grid is outlined by Varaiya et al. [44]. In contrast to their work, our focus is to leverage SGAM modeling [39] to develop a practical risk assessment approach usable for utilities on top of a holistic smart grid implementation model with a current to near future perspective. Smart grid risk assessment is only one step in the cyclic smart grid security management process. We extended the state-of-the-art with our architecture model driven risk assessment (Chapter 2) and management approach (Chapter 3). A more detailed overview of the state-of-the-art within these topics is provided in the corresponding Sections 2.1 and 3.1 of these chapters.

With architecture model driven smart grid risk assessment, high risk components can be identified and selected for embedded security analysis. However, the firmware of these devices is often not available to independent security analysts as the firmware is locked in microchips and can not be read out in the first place. In Chapter 4, we explore physical attacks on smart grid devices in general and describe several attacks that can also be used to dump the firmware of smart grid embedded devices for subsequent security analysis. Weingart et al. [123] provide a survey of physical attacks and defenses. Skorobogatov [101] outlines active fault injection and glitching attacks while Kocher first introduced passive side channel [63] and power analysis attacks [64]. The idea of using different side channels to carry out attacks has been extended to EM emission attacks by researchers such as Agrawal et al. [1]. The concept of using side channel information for template attacks has been further explored by Chari et al. [19]. While previous work describes many of these practical attacks, we describe the use of these attacks specifically for critical infrastructure and smart grid devices. Some of the described attacks can not only be used to attack the embedded devices themselves, but they are also usable for independent security analysts to obtain the firmware for subsequent security testing.

In Chapter 5, we explored the idea of leveraging physical attacks for firmware extraction further and show that limited IC reverse engineering can be utilized to discover manufacturing test modes allowing to dump the device firmware. Several approaches to investigate IC test modes have been explored by researchers, but most of them are non-invasive in nature. For instance, Sergei Skorobogatov and Christopher Woods explore undocumented JTAG features using side-channel measurements [100]. In [24], Jean Da Rolt et al. demonstrate an attack on single and multiple scan chain structures to obtain secret AES encryption keys hidden within the chip. In [42], David Hély et al. suggest scan chain scrambling against these non-invasive attacks. In contrast to the described work, our approach is based on an invasive attack and not impeded by the suggested countermeasures. Invasive IC reverse engineering was demonstrated by researchers such as Karsten Nohl et al. as well [82]. Although we use a similar method, we specifically explore limited effort IC reverse engineering to target IC testing logic for firmware extraction.

Once the firmware is available to the analyst, it can be analyzed to identify security vulnerabilities. For embedded firmware security analysis, current research approaches mostly focus on static analysis to achieve their goal. Khare et al. highlight key problems when using static analysis techniques on a large embedded code base [59]. In contrast to our work, they perform the analysis techniques on source code which is frequently not available to the analyst. Instead, our approach is applied to the binary firmware image and access to source code is not required. Ramakrishnan and Gopal do not require access to the source code as well as their static program analysis techniques run on embedded binaries [118]. However, their focus is not on embedded firmware security analysis or vulnerability discovery. Zili Shao et al. describe a framework for embedded systems to check for and protect from buffer overflow attacks [96]. Their system is more focused on vulnerability protection than on vulnerability discovery. In [108], Sumpf and Brakensiek introduce device driver isolation within virtualized embedded platforms. Their approach is somehow related to our dynamic firmware security testing approach although their application is vastly different. In Chapter 6, we present PROSPECT, a system enabling dynamic security testing on embedded system firmware. Our work on PROSPECT was published in 2014 [56]. In the same year, Zaddach et al. presented the Avatar framework [134] that also addresses the dynamic firmware security testing problem. While our work relies on an embedded operating system kernel, Avatar works on a lower level by cleverly instrumenting existing tools such as the QEMU emulator.

Considering systems such as Avatar [134] or PROSPECT, we explored the use of peripheral device communication caching in combination with program state approximation. Although to the best of our knowledge we are the first to use this approach to support embedded firmware security testing, the concept underlying our caching heuristic is related to the well known problem of program slicing. Program slicing typically works on source code and has been broadly covered by Weiser et al. [124], Korel et al. [66], Frank Tip [110] and Binkley et al. [8]. The problem of binary program slicing with no access to source code has been covered more recently by Kiss et al. [61] and Cifuentes et al. [21]. In contrast to exact program slicing, our caching approach relies on weak program state approximation instead and is presented in Chapter 7.

To cover the practical analysis of cryptographic authentication mechanisms and protocols as well, we explore FPGA clustering supported high speed attacks in Chapter 8 and present a high-speed attack on the well known WPA2 Personal cryptographic authentication protocol. Researchers such as Johnson et al. [48] have presented similar attacks. In comparison to their mostly sequential design, we use a full pipelined approach that leads to significantly higher performance. Güneysu et al. present the RIVYERA and COPACOBANA high-performance FPGA cluster systems for cryptanalysis [40]. They cover a wide range practical attacks on algorithms such as DES, Hitag2 or Keeloq and have a larger cluster configuration than we had available for our tests. In contrast, our work relies on repurposed cryptocurrency mining FPGA systems and on the well known WPA2-Personal cryptographic protocol instead. We also compare our system to existing commercial FPGA cluster systems [32, 89] and claim that on the same hardware our implementation is significantly faster than the world's currently fastest commercial FPGA-based password cracking solution [31].

1.4 Methodological Approach

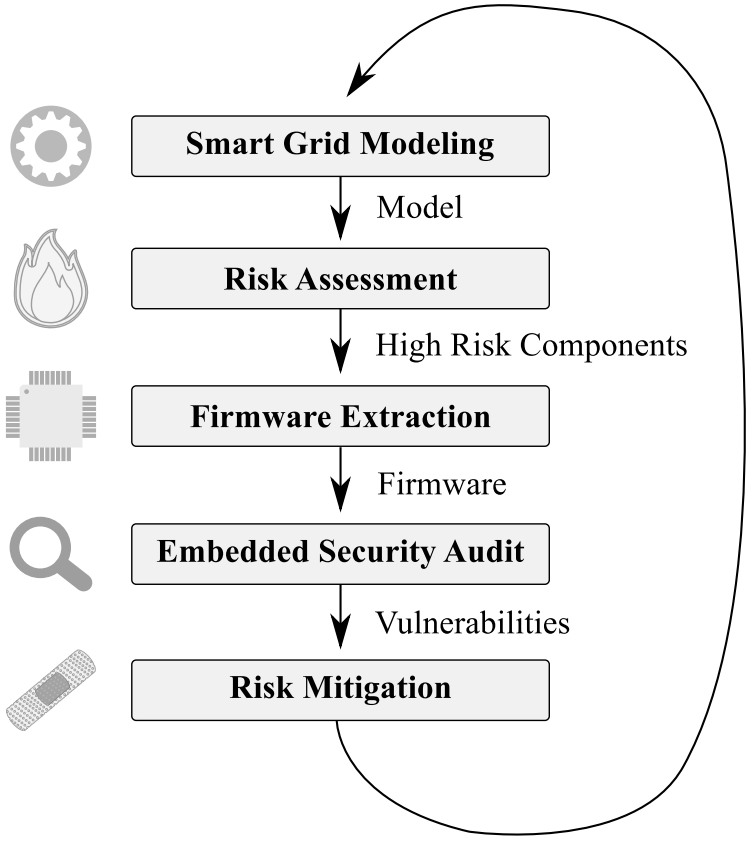

Figure 1.2: Holistic Smart Grid Risk Management Process

Our overall methodological approach is visible in Figure 1.2. In the first step, we use SGAM modeling [39] on top of both national and international smart grid projects that cover either currently rolled out or near future technologies. The aim of this approach is to create a cumulative model that reflects current and near future smart grid technologies in European smart grids. We ensure that this model is realistic through feedback rounds with utilities and manufacturers. In the second step, we evaluate well respected classical [35] and smart grid centric security management processes [34, 36, 37, 84, 85] in feedback rounds with leading national utilities and manufacturers. The aim is to determine their practical usability, how well those standards cover the actual architectures in the field and the reasons why they are not utilized by utilities. Based on threat sources such as [35–37] and [84], we develop a threat catalog by evaluating a large number of threats from a smart grid perspective. By applying the threat catalog to the cumulative SGAM model, we create a threat matrix that allows users (such as utilities) to assess the risk potential for smart grid components on an architectural level. At this point, smart grid components with the highest risk potentials are identified using the threat matrix and selected for a practical embedded security analysis in a laboratory setup. In the third step, we explore physical attacks on smart

grid devices and show that some of these attacks can be utilized to extract the firmware from embedded smart grid devices. If common firmware extraction approaches such as the use of debug and programming interfaces fail (e.g., either due to not being available or due to security fuse configuration preventing firmware readout), we amend the problem with limited effort test mode silicon reverse engineering. We present a case study showing that even proprietary chip test modes can be reverse engineered with limited effort and utilized to extract the device firmware. Once the firmware is available to the analyst, in the fourth step, we address the shortcomings of prevalent static analysis approaches in embedded security analysis [13, 59, 118]. We investigate the use of emulators with proprietary peripheral device communication forwarding to enable dynamic security analysis approaches such as fuzz testing [6, 70] and introduce PROSPECT, a proxy capable of tunneling arbitrary peripheral hardware accesses from within a virtual machine to the embedded system under test. The result is a virtualized execution environment for embedded software implementations with a completely transparent connection to the actual peripheral hardware components of the system under test. PROSPECT thus enables the analysts to leverage any powerful dynamic analysis techniques of their choice [6, 17, 70, 94] to discover vulnerabilities on embedded devices with minimal effort. Based on PROSPECT, we also explore the use of peripheral device caching and utilize firmware program state approximation to determine whether a device is already in the cache and, if so, which of the cached device responses need to be returned to the firmware under test. Our program state approximation heuristic is related to program slicing [21, 61] and, similar to symbolic execution, suffers from the well known state explosion problem [94]. Finally, to outline highly efficient attacks on future cryptographic smart grid authentication protocols, we present the design and implementation of an FPGA (Field Programmable Gate Array) cluster based high-speed brute-force attack on the well known WPA-2 Personal authentication protocol. In the last step, based on the obtained results, we finally close the cyclic smart grid security management process and show existing security problems and risk mitigation strategies for current and near future smart grids.
Throughout the research conducted during this thesis, it has been highly challenging to publish or otherwise publicly disclose results concerning smart grid security. There were non-disclosure agreements in place and results we obtained were often considered to impact national security. To create scientific publications in spite of these challenges, we chose to evaluate some of the approaches presented in this thesis on less critical embedded devices and protocols outside of the smart grid domain instead.

1.5 Structure of this Work

This thesis is organized as follows: Chapter 2 presents a practical smart grid risk assessment approach that utilizes a smart grid model based on the European Smart Grid Reference Architecture [103]. The approach allows smart grid operators such as utilities to better identify high risk smart grid systems that should undergo further security analysis. This work has been published under the same title at the *3rd International Conference on Smart Grids and Green IT Systems (SMARTGREENS)* in 2014 [52] and was carried out in collaboration with *Lucie Langer*, *Florian Skopik*, *Friederich Kupzog* and *Wolfgang Kastner*. Chapter 3 extends the approach of utilizing a cumulative smart grid model and presents a holistic architecture-driven security management

approach for smart grids. The approach is realized in a cyclic process including a modeling step, risk identification, risk assessment through security testing on embedded smart grid field components as well as risk mitigation and compliance checking steps. The approach has been published under the title "Architecture-driven smart grid security management" at the *2nd ACM Workshop on Information Hiding and Multimedia Security* in 2014 [51] conducted in collaboration with *Lucie Langer*, *Florian Skopik* and *Wolfgang Kastner*. Chapter 4 presents an overview of physical attacks on embedded smart grid devices. While the described physical attacks have a broader scope than just firmware extraction, many of them can support independent security analysts to extract and subsequently analyze device firmware. At the same time, manufacturers can increase smart grid device security by considering these physical attacks in their design. The chapter presented in this thesis is a subset of the chapter "Resilience against physical attacks" that has been published in collaboration with *Martin Hutle* in the book *Smart Grid Security: Innovative Solutions for a Modernized Grid* in 2015 [99]. From these physical attacks, Chapter 5 explores the use of invasive limited-effort semiconductor reverse engineering to uncover the chip-internal secrets of commonly utilized silicon test modes. Once the included test modes are understood, it is often possible to extract deeply hidden firmware from the chip and subsequently analyze it with respect to its security properties. Since smart grid microchips were unavailable for this kind of analysis and the results would have been under strict non-disclosure agreements, we conducted a case study on a cryptographic game console authentication chip instead. This work has been published under the title "Breaking Integrated Circuit Device Security through Test Mode Silicon Reverse Engineering" at the *21st ACM Conference on Computer and Communications Security (ACM CCS)* in 2014 [54] in collaboration with with my colleagues *Markus Muellner*, *Daniel Burian*, *Christian Platzer* and *Wolfgang Kastner*. Chapter 6 introduces PROSPECT, a system that addresses the problem of peripheral device communication during the dynamic firmware analysis inside a virtualized analysis environment. PROSPECT (Peripheral Proxying Supported Embedded Code Testing) leverages peripheral device communication forwarding so that the firmware under analysis can be executed inside a virtualized environment even though the actual peripheral devices are located on the real embedded platform instead. PROSPECT has been published under the title "PROSPECT: peripheral proxying supported embedded code testing" at the *9th ACM Symposium on Information, Computer and Communications Security (ASIACCS)* in 2014 [56] in collaboration with my colleagues *Christian Platzer* and *Wolfgang Kastner*. Chapter 7 takes the idea of PROSPECT further and introduces a peripheral device communication cache. During security testing techniques such as fuzz testing it is common that the same firmware code regions are executed over and over again. Since peripheral device communication triggered by these code regions will be highly identical, the approach would allow the caching system to be trained so that subsequent test cases can be conducted with device responses from the cache. As a result, existing dynamic firmware analysis techniques could become more powerful by enabling functions such as snapshotting, test parallelization or testing without physical access to the embedded system. The feasibility study has been conducted in collaboration with *Daniel Burian* and *Wolfgang Kastner* and was published under the title "Embedded Security Testing with Peripheral Device Caching and Runtime Program State Approximation" at the *10th International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)* in 2016 [50]. While our work in the smart grid security domain has shown that many smart grid

devices still communicate over insecure unencrypted protocols, more secure communication and authentication mechanisms leveraging cryptographic primitives will be on the rise in the smart grid domain as well. To provide an outlook in efficient testing of those protocols, in Chapter 8 we explore the use of special purpose cryptographic hardware based on FPGA devices to attack cryptographic authentication protocols. Due to a general lack of strong cryptographic authentication protocols in current smart grid systems, in this work we present a very powerful hardware assisted brute-force attack implementation on the widely established WPA-2 Personal authentication protocol instead. Our work shows that FPGAs and FPGA clusters can be very effectively used to attack cryptographic authentication mechanisms and, as a result, future smart grid authentication protocols need to consider these attacks just the same. This work has been published without the conducted case study under the title "Efficient High-Speed WPA2 Brute Force Attacks using Scalable Low-Cost FPGA Clustering" at the *Workshop on Cryptographic Hardware and Embedded Systems (CHES)* in 2016 [55] in collaboration with my colleagues *Markus Muellner*, *Christian Kudera* and *Wolfgang Kastner*.

1.6 Summary of this Work

Smart Grid Modeling

In Chapters 2 and 3, we present a practical smart grid architecture model driven risk assessment and security management approach that has been published under the title "Practical Risk Assessment Using a Cumulative Smart Grid Model" at the *3rd International Conference on Smart Grids and Green IT Systems (SMARTGREENS)* in 2014 [52] and under the title "Architecture-driven smart grid security management" at the *2nd ACM Workshop on Information Hiding and Multimedia Security* in 2014 [51]. In our joint approach to identify security risks in current and near future smart grids, it turned out that smart grid experts, utilities and even manufacturers have a very different view of what the smart grid is. Without a common view it is however infeasible to leverage the combined knowledge for risk assessment. Consequently in the first step, we asked utilities to compare their deployed smart grid systems with the existing European Smart Grid Reference Architecture [103]. Specifically, we used the Smart Grid Architecture Model (SGAM) [103] to allow for a well-structured comparison of 45 different smart grid projects prioritized according to project size, project relevance, and both amount and quality of available information. SGAM modeling initially originated from the need to identify gaps in standardization, but today it serves three major purposes: It is a means to visualize and compare different smart grid architectures, is allows to identify gaps in all layers and, finally, SGAM can serve as a useful model to support model-driven architecture development. An overview of the model is visible in Figure 1.3.

The model comprises *zones*, *domains* and *interoperability layers*. The zones are derived from the typical layers of a hierarchical automation system [93], while the domains reflect power-system specific fields of different actors. In the third dimension, the SGAM interoperability layers covering the different aspects of networked smart grid systems are aligned. The study showed that the reference model is only applicable on a high level and it lacks the detailed technological information that would be required as a basis for qualified risk modeling. To close this gap,

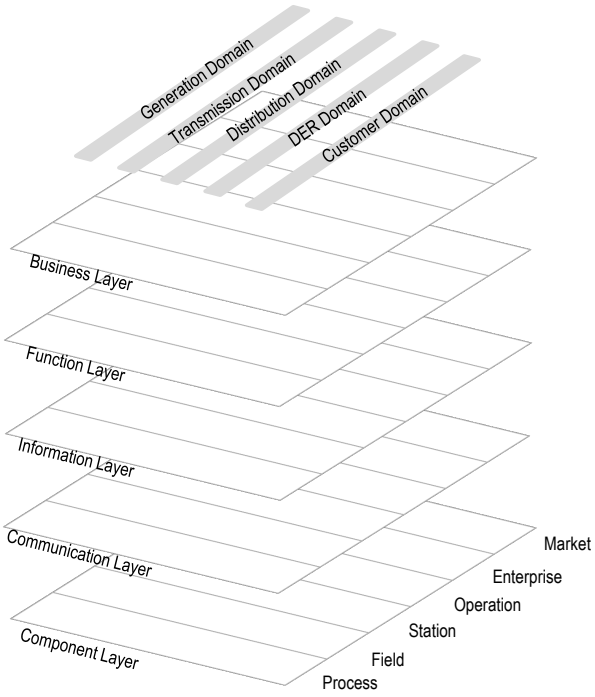

Figure 1.3: Smart Grid Architecture Model (SGAM) Framework

together with utilities and manufacturers, we combined six national and four international projects within the SGAM framework to form a cumulative architecture model that represents current and future smart grid installations in Europe. For SGAM modeling in addition to the project significance, detailed availability of information about the technical implementations was an additional selection criterion for the selected projects. The following significant national smart grid research projects were selected for modeling:

- IEM: Intelligent Energy Management
- Smart Web Grids [105]
- DG DemoNetz Smart LV Grids [62]
- ZUQDE: Zentrale Spannungs- und Blindleistungsregelung mit dezentralen Einspeisungen in der Demoregion Salzburg [104]
- EMPORA: E-Mobile Power Austria [29]
- AMIS Smart Metering Rollout

A similar approach was applied to international projects. The selected significant projects were:

- The European FP7 Project OpenNode [87]
- The European FP7 Project EcoGrid EU [30]
- The US Demand Response Automation Server (DRAS) [26]
- The German ICT Gateway Approach OGEMA [86]

The resulting architecture model (Fig. 1.4) includes a harmonized cumulative view of the components found in all analyzed projects and shows the component and communication layer of the SGAM framework. We evaluated the architecture model in a number of feedback rounds with utilities and manufacturers leading to the integration of subsequent improvements and additional technical information. The feedback rounds thus also ensured that the model realistically reflects the utilities' existing smart grid installations as well. On the bottom, the field devices (such as smart meters or dedicated sensors and actuators) can be found while on the station level, both primary and secondary substation including their respective automation components are located. The model also includes the customer side with residential customers, commercial buildings, and electric mobility. At the top, the enterprise level and the marked components are visible.
While this model specifically represents current and near future smart grids in Europe, the major difference to existing models is the inclusion of detailed communication technologies allowing a more in-depth risk assessment approach. The underlined communication protocols and technologies are the ones that are predominantly used in the projects we analyzed. Regarding the subsequent risk assessment approach, for instance, an insecure directional wireless communication link is much more likely to be targeted by attackers in comparison to a cryptographically secured wired fiber link. Since the architecture model includes the necessary information on employed communication protocols and technologies as well, model driven smart grid risk assessment becomes feasible. The cumulative smart grid architecture model thus serves as an anchor point for further analysis and as a common document of energy, IT and security experts. Since the release and publication of the model at the *3rd International Conference on Smart Grids and Green IT Systems (SMARTGREENS)* (Chapter 2) and the *2nd ACM Workshop on Information Hiding and Multimedia Security (IHMMSEC)* (Chapter 3), the model and the subsequent risk assessment approach have been utilized by national utilities and the Austrian regulator E-Control [28].

Smart Grid Risk Assessment

Smart grid risk assessment has been addressed in several several standards, guidelines and recommendations. We analyzed the existing approaches to define a practical risk assessment approach on top of the cumulative smart grid model (Figure 1.4). The U.S. National Institute of Standards and Technology (NIST) has issued a report on "Guidelines for Smart Grid Cyber Security (NIST-IR 7628)" [81]. It primarily focuses on customer privacy related risks and high-level risk mitigation strategies but no general approach for assessing security risks in the smart grid is provided. Based on existing work such as NIST-IR 7628 and ISO 27002, the European Network and Information Security Agency (ENISA) has issued a report on smart grid security

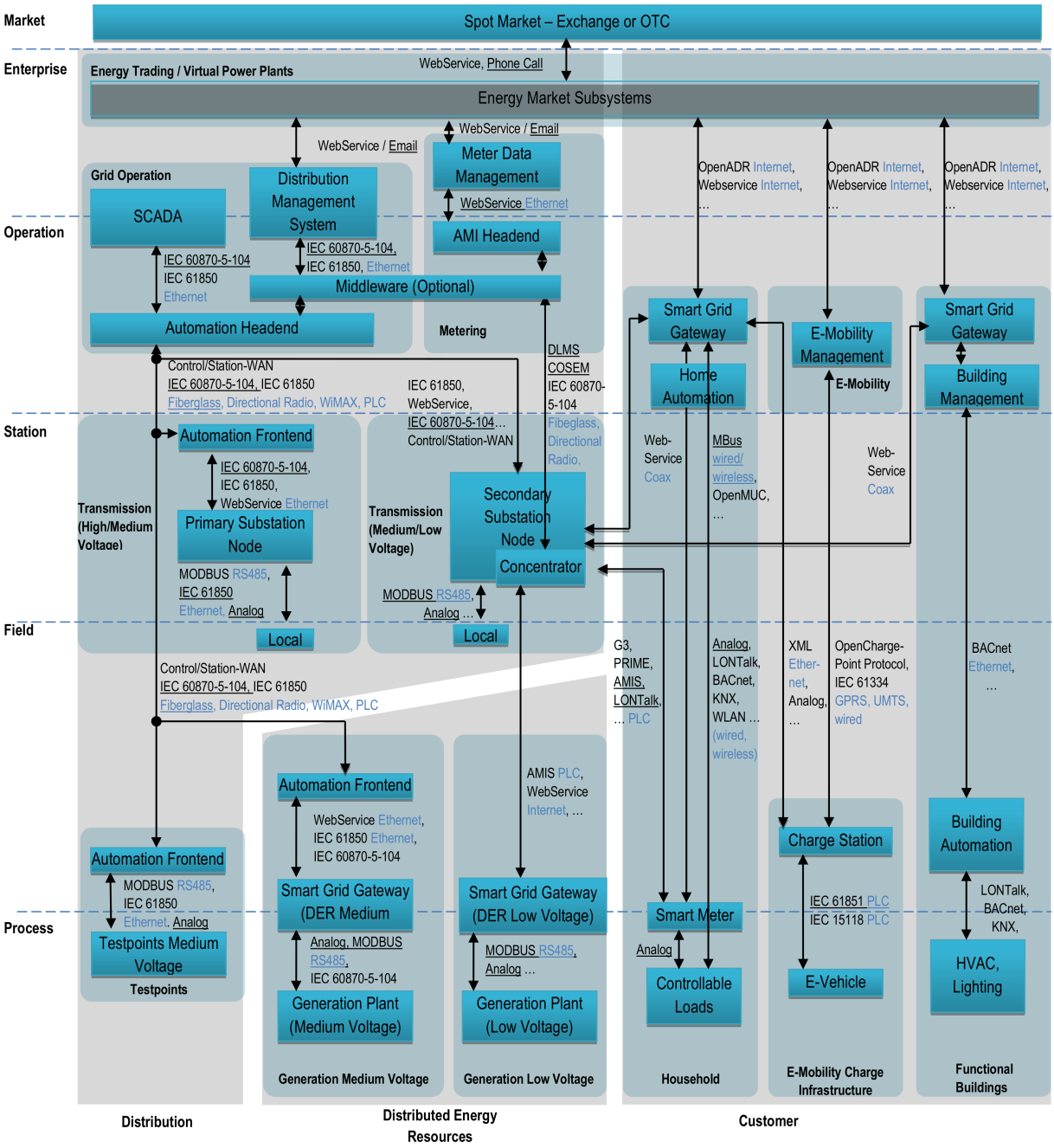

Figure 1.4: Cumulative Smart Grid Model

to establish a minimum set of security in European smart grids [33]. It defines several security measures categorized into three different "sophistication levels" ranging from early-stage to advanced ones. The report specifically mentions that proper risk assessment needs to be carried out to determine the required sophistication levels, but no actual risk assessment methodology is identified within the report. Besides NIST and ENISA, the German Federal Office for Information Security (BSI) has come up with Common Criteria Protection Profiles [15, 16]. The protection profiles are focused on smart meters only (as visible in Figure 1.4, smart metering is just one of the many components in a smart grid) and define the minimum security requirements for these devices based on a threat analysis. However, due to the smart metering focus of the Common Criteria approach, it cannot provide a holistic view on cyber security threats in future smart grids. Another drawback is that the protection profiles focus on the German market whereas many smart metering systems currently being rolled out in European countries are vastly different from the BSI concept and do not correspond to the defined gateway and security module design concept. Among other reports, the CEN-CENELEC-ETSI Smart Grid Coordination Group released the "Smart Grid Information Security (SGIS)" report as a response to the EU Smart Grid Mandate M/490 [102]. The report defines five SGIS Security Levels which are used to assess the criticality of smart grid components based on power loss and ICT system failures. However, since the risk assessment is based on the assumption that no security controls are in place, it is not suitable for current installations or foreseeable implementations that already include at least some level security. The FP7 project EURACOM considered protection and resilience of energy supply in Europe with the similar aim to identify a holistic approach for risk assessment within the energy sector. Their methodology is however asset-driven while our approach is required to be architecture-driven based on our architecture model. We developed our architecture-driven risk assessment approach with a focus on smart grids in Europe from a Distribution System Operator's perspective through the following steps:

1. Compile a *threat catalog* for smart grids focusing on ICT-related threats and vulnerabilities
2. Develop a *threat matrix* by applying the threat catalog to the ICT architecture model, i.e., identify which threats apply to which components of the model
3. Assess the potential risk for each element within the threat matrix by estimating the probability and the impact of an according attack, thus eventually producing a *risk catalog*

Since our threat catalog should not be developed from scratch but instead builds upon well-established sources of ICT-related security threats, we focused on the IT Baseline Protection Catalogs developed by the Federal Office for Information Security [14] as our main source. In addition, we included the threats specified in the smart-grid-specific Protection Profiles [15, 16]. Altogether we collected a set of initially 500 threats accumulated from the identified sources which we carefully analyzed for their relevance for the smart grid and the ICT components it relies on by adapting the threats to the smart grid scenario (i.e., they were interpreted in the smart grid context). This step resulted in roughly half of the initial threats for further consideration. Since some of these threats were highly specific while others were on a rather high level, we performed a "weeding" step to obtain an equal threat granularity.

The result is a list of 31 threats grouped into the following clusters for practical threat modeling:

- Authentification / Authorization
- Cryptography / Confidentiality
- Integrity / Availability
- Missing / Inadequate Security Controls
- Internal / External Interfaces
- Maintenance / System Status

The next step was to apply the threat catalog to the cumulative smart grid model by analyzing which threats are relevant for which of the modeled components and why. We assessed the functionality and the characteristics of the individual architecture components at the granularity level of the domains depicted in dark gray in Figure 1.4:

- Functional Buildings
- E-Mobility & Charge Infrastructure
- Household
- Generation Low Voltage
- Generation Medium Voltage
- Testpoints
- Transmission (High/Medium Voltage)
- Transmission (Medium/Low Voltage)
- Grid Operation
- Metering

We did not consider the Energy Markets domain due to the lack of current ICT utilization and the lack of information on future functionalities. For each element in our threat matrix, we developed potential attack scenarios and analyzed whether the specific threat could be relevant. We documented the decision process and verified it through multiple feedback rounds with manufacturers and utilities.
In the third step, we assessed the risk potential on the components in the cumulative smart grid model with a semi-quantitative approach based on the probability and the impact of each of the threats. On a five-level scale ranging from very low (level 1) to very high (level 5), the probability level was determined by the number of successful attacks per year (less than 0.1 incidents (level 1) to multiple incidents (level 5) per year). Similarly, the impact of a successful attack was

determined by the level of the local, regional and global impact such as monetary loss, customer impact or geographic range. The outcome of this step is our risk catalog, a comprehensive catalog of cyber security risks on smart grids in Europe. To ensure practical usability and realistic results, we evaluated our approach utilizing a three step evaluation methodology:

- *Step 1: Evaluation of Threat Catalog.* We set up end user workshops with experts from utility providers, device manufacturers and academic institutions to evaluate both the relevance and completeness of the identified threats.
- *Step 2: Threat Relevance.* In similar workshops, experts surveyed the applicability of identified threats to the various domains in the SGAM framework. The result was again discussed and refined with major utility providers in Austria in end-user centric workshops.
- *Step 3: Probability and Impact Assessment.* In a last step, we had experts independently rate the probability of occurrence of identified threats to include the opinions of different utility providers. The individual results were again discussed and consolidated to ensure the broad use of the resulting threat and risk catalog.

In addition to the development of the practical architecture model driven risk assessment approach, we identified the following four major findings:

Unbalanced Risk Distribution The risk distribution in smart grids is unbalanced. Although attacks on the lower levels of the cumulative smart grid model are more likely due to easier access, the impact is significantly less severe. On the other hand, attacks on the upper levels are much harder to carry out, but if successful, their impact can be disastrous.

Evolution of the Grid Today, both legacy systems without ICT technologies as well as modern systems need to be interoperable. As a result we need to face many security challenges including: (i) the mix of legacy protocols and new protocols; (ii) the usage of wrappers, data converters, and gateways to make devices interoperable, and (iii) short innovation cycles where technologies advance in a clash with the traditional views of grid investments with components designed to last for decades.

Technological Diversity Our analysis of smart grid projects shows that there is a high grade of technological diversity due to the different employed technologies [98]. The diversity and the need for seamless interoperability mostly avoids rigid designs and the setup of a uniform and secure architecture, but at the same time it can also prevent cascading effects as the potential vulnerabilities in the system of one device and/or system manufacturer are typically different from another manufacturer.

Risk Assessment Complexity In contrast to traditional components in power grids, smart grids heavily rely on the introduction of ICT components that significantly raise the complexity of risk assessment. Even across geographical borders, systems become more and more interdependent on each other leading to the advent of potential large scale and complex vulnerabilities. Estimating

risks for such cases is extremely difficult, since numerous mostly unknown variables need to be considered across many communication links and systems. We argue that with our architecture model driven risk assessment approach, the existing communication links as well as utilized protocols are clearly documented and their interdependencies are clearly visible. As a result, we believe that our approach can be of significant help to address the risk assessment complexity.

Physical Attacks and Firmware Extraction

Once high risk components have been identified through smart grid risk assessment, the next step is to conduct an embedded security audit on these components. In Chapter 4, we explore physical attacks on embedded smart grid components that have been published in the book chapter "Resilience against physical attacks" in the book *Smart Grid Security: Innovative Solutions for a Modernized Grid* in 2015 by Elsevier Science Publishing [99]. On a high level, physical attacks can serve two purposes: First, security critical smart grid devices like smart meters are often physically located in areas such as private households that are easy to access by potential attackers. Due to their accessibility, an attacker could thus mount physical attacks to manipulate a device such as a smart meter to gain an advantage (e.g., lower energy costs) or to extract secret cryptographic key material for subsequent attacks on the utility's ICT network infrastructure. However second, physical attacks can also aid independent security researchers to conduct security audits. For instance, the majority of the security critical device implementations such as authentication or protocol handling resides in the device's firmware. As a result, the firmware needs to be analyzed for security vulnerabilities, but it might not be available to the independent security researcher in the first place. For instance, the firmware could be stored in a non-volatile memory chip or within a microcontroller with a security fuse to prevent firmware readout. In these cases, security researchers can turn to physical attacks to dump the firmware or to analyze security critical functions in the hardware itself. In Chapter 4, we initially present the goals of physical attacks on smart grid devices. These goals are either passive *information gathering* or active *device manipulation*. As physical attacks typically require lab testing equipment to carry them out, we provide an overview of the typically required equipment, the equipment costs and the resulting attack potential in Table 1.1.
While many attacks such as accessing or probing open interfaces are possible with equipment available for less than EUR 1,000, more advanced attacks also require more expensive lab equipment with an increasing level of sophistication. One of the major reasons is the ongoing high integration depth of electronic components. While in the past, components such as memories, CPUs and peripheral devices were all distributed across Printed Circuit Boards (PCBs) and thus easily accessible, modern devices such as microcontrollers have these components integrated within the silicon chip. As result, an attacker can no longer easily probe the interconnections of the electronic components located on the PCB, but depending on the type of attack it might be necessary to remove the package of an integrated circuit and probe nanometer-wide traces instead. In general, we thus categorize physical attacks in non-invasive, semi-invasive and invasive attacks. A taxonomic overview of these attack classes is provided in Figure 1.5. Non-invasive attacks are attacks that can be carried out without having to open the package of integrated circuits. For instance, an attacker could try to access a local bus to read and/or modify signals exchanged between digital electronic components on the PCB. With semi-invasive attacks, the package of

Attack potential	Techniques	Typical equipment	Typical equipment costs (EUR)
Low	Access to local storage Accessing open interfaces Probing on buses Simple faults	memory chip reader, logic analyzer, microcontroller/FPGA boards	less than 1,000
Medium	Simple side channel attacks Glitching Attacks	digital oscilloscope, signal generator, FPGA boards	2,000 - 10,000
Elevated	Enhanced side channel attacks EMA DPA Template attacks Semi-invasive attacks	high-resolution digital oscilloscope, FPGA boards, chemical depackaging	10,000 - 50,000
High	Invasive attacks Fault Attacks	(laser) probing station, chemical depackaging, focused ion beam (FIB)	more than 50,000

Table 1.1: Overview of the Physical Attack Potential

the integrated circuit is opened typically with wet chemical etching processes, but the silicon die within the package is not modified in any way. Since the die is clearly visible now, the attacker can use semi-invasive attacks such as optical attacks to either passively analyze or to actively change functions on the chip during runtime. Invasive attacks go even further by modifying the silicon die itself to enable attacks such as probing on internal buses or reverse engineering of logic blocks implemented in silicon.
For each of those attack classes, we describe practical physical attacks on smart grid devices in Section 4.2. The non-invasive attacks include access to local storage, accessing open interfaces, bus probing, fault and signal injection as well as side-channel attacks. For instance, access to local storage is an attack that can be carried out at low costs if the embedded device utilizes external non-voltage memory chips such as NAND or NOR flash. In this case, the attacker or the analyst can unsolder the memory chip and connect it to a low-cost microcontroller or FPGA board to read out the firmware. An example of this approach is visible in Figure 1.6 that shows how the firmware has been extracted from a smart meter device.
Embedded smart grid devices typically include numerous interfaces to communicate with the outside world. In addition, the circuit boards of these devices often include debug and programming ports such as JTAG (Joint Test Action Group) or serial debug consoles as well. Especially programming ports such as JTAG are extremely powerful as they can not only be used for programming, but they can often be utilized for firmware readout or dynamic firmware debugging just the same. Attackers or analysts can thus utilize these open interfaces to read out and/or debug the firmware or to change firmware behavior.

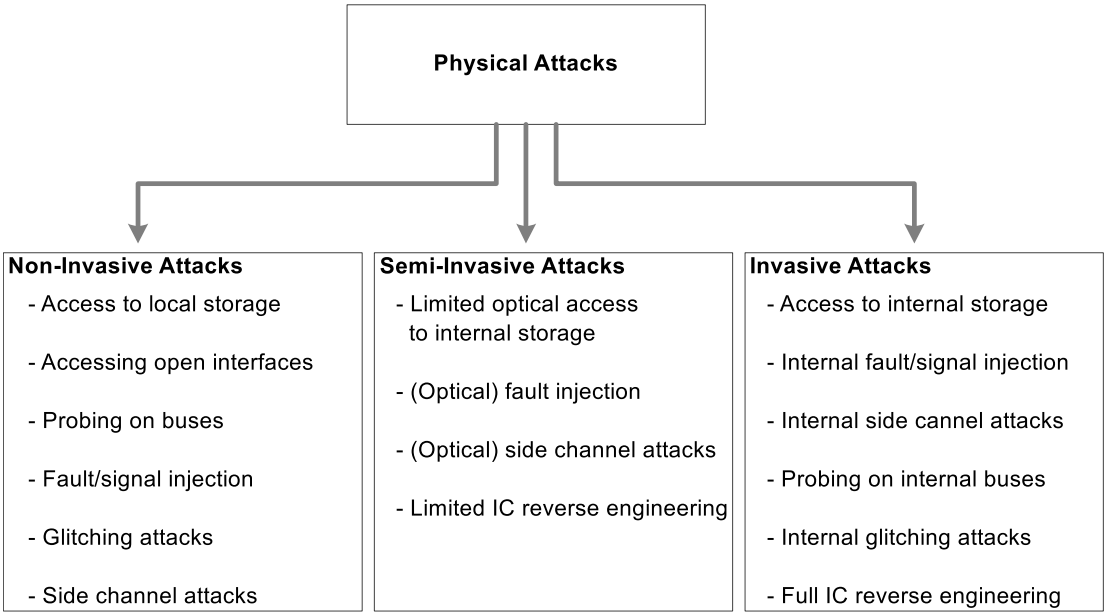

Figure 1.5: Taxonomy of Physical Attacks

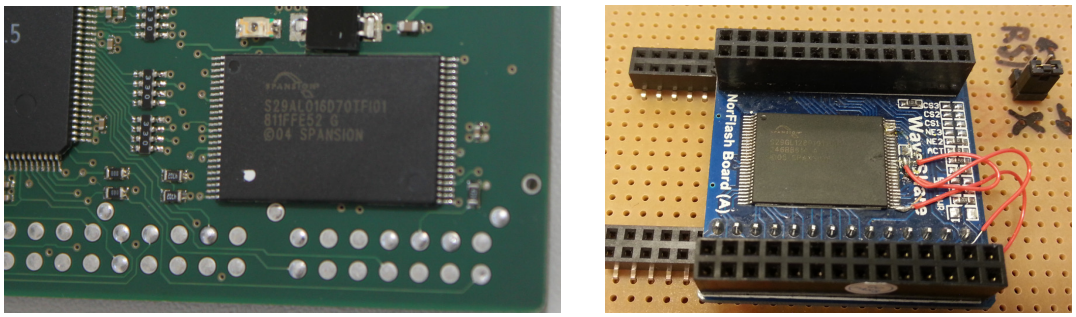

Figure 1.6: Flash Memory Chip in a Smart Meter (left), Smart Meter Memory Chip Soldered to Breakout Board for Readout (right)

Figure 1.7 shows identified programming ports in a smart grid substation automation system and a smart meter. At the example of the smart meter it is visible how a simple breakout board has been soldered to access the programming port and connect it to a JTAG debug dongle. With this setup, we were able to read out and subsequently debug the smart meter firmware.

Local bus probing is another easy to carry out physical attack to gather information on inter-device communication on the circuit board. For instance, a microcontroller chip within an embedded system could utilize a standard serial bus such as SPI or I2C to communicate with a communication module such as a GSM modem. By utilizing standard measurement equipment such as an oscilloscope or a logic analyzer, an attacker could probe the bus between those components and analyze any information exchanged between them. As a result, the attacker could learn the commands that are exchanged over the bus and use them for subsequent attacks. For instance,

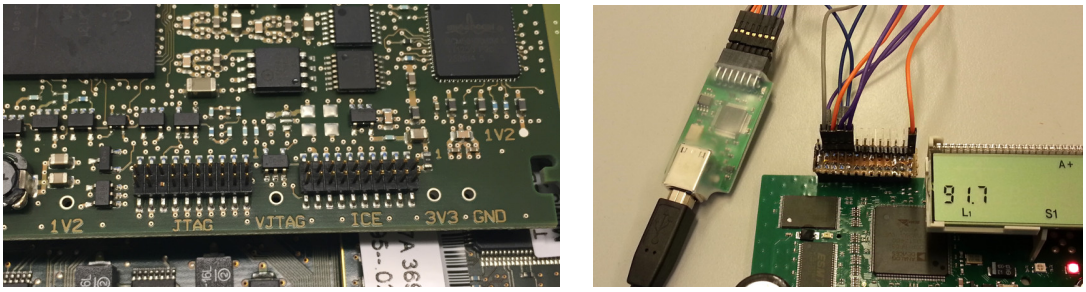

Figure 1.7: JTAG and In-Circuit Emulation (ICE) Ports Inside a Substation Automation System (left), Connected JTAG Interface on a Smart Meter (right)

the attacker could disconnect the communication module and instead connect a microcontroller board with his own firmware. The firmware would subsequently pose as communication module and could potentially return commands or data that would trigger unwanted behavior. However, an attack targeted towards the communication module could probably have a high impact. In that case, the attacker would disconnect the original microcontroller and connect his own one to the communication module. This would allow him to send arbitrary commands over the communication module to the utility on the other side.
If the firmware residing within an external non-volatile memory chip is encrypted or it is inaccessible (e.g., due to the integration of the non-volatile memory in the controller chip itself), an attacker can resort to side channel and fault injection attacks. Considering a typical controller chip, the general idea of a side channel attack is to passively measure variations in the operating environment such as current draw, electromagnetic emanation or response time during operation. Depending on the operations and the data the chip processes, these parameters change significantly if observed on a small enough time scale. An attacker can thus leverage side channel information to gather information on the internally processed instructions and data if the implementation is not protected against these kinds of attacks. For instance, considering the above example of an encrypted external memory chip, the attacker could attempt to measure the current draw of the controller during decryption to obtain the secret decryption key or the secret password to the chip's bootloader (power analysis attack). In Section 4.2, we describe timing attacks, power analysis attacks including Simple Power Analysis (SPA), Differential Power Analysis (DPA) and template attacks as well as attacks utilizing electromagnetic emanation (EM).

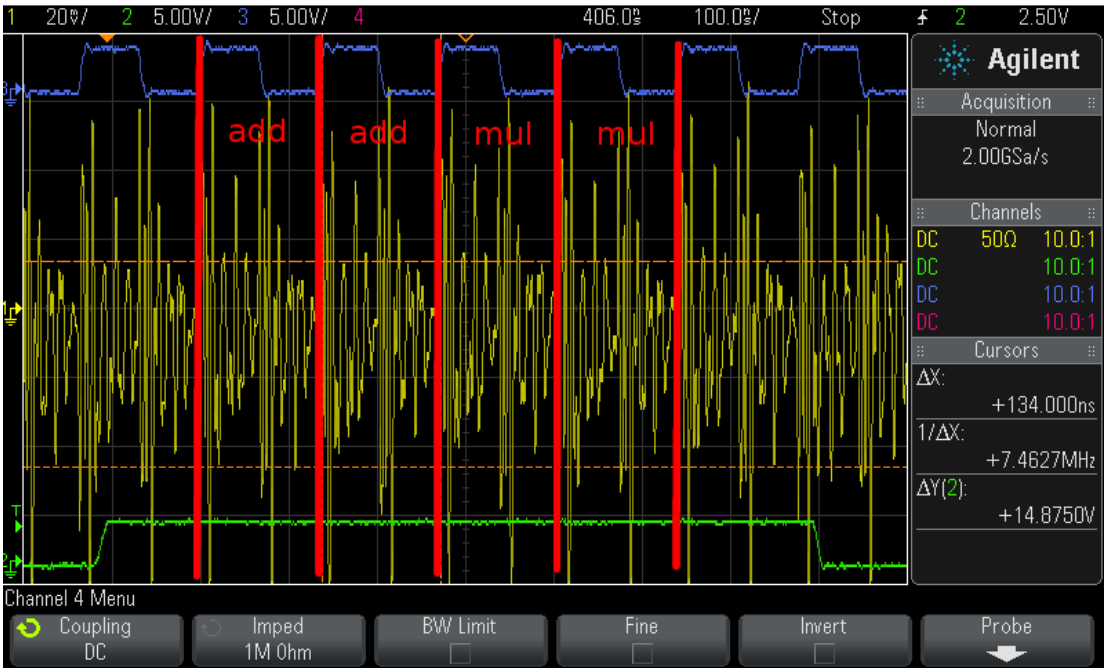

Figure 1.8: Exemplary Power Trace of a Microcontroller During the Execution of a Simple Algorithm

With SPA the attacker directly analyzes the taken measurements to get clues on executed instructions and data (Figure 1.8). Since this approach is somewhat limited and often tedious, DPA uses a different approach. The general idea of DPA attacks is to use a power model. The attacker creates hypotheses about the processed data (i.e., one byte of the encryption key) and computes the theoretic power trace for each of the hypotheses. These theoretic power traces are then compared to the actual measured power traces by computing the statistical correlation. Due to the use of statistical correlation, DPA is also known as Correlation Power Analysis (CPA). The higher the correlation, the higher is the match between the measured power trace and the power trace hypothesis. This way the attacker can remove implausible candidates and ultimately determine the correct guess.
An exemplary setup to carry out a DPA attack is visible in Figure 1.9. Template attacks and other profiling attacks assume that the attacker has full control over an identical device. The device under control of the attacker is used to create a power model of interesting operations such as symmetric or asymmetric cryptographic routines. The model is then applied on the other device to identify these operations and the processed data. EM attacks use electromagnetic emanation instead of power analysis. While the same types of attacks as with traditional power analysis can be applied (i.e., SPA, DPA and template attacks), the advantage of EM attacks is that in comparison much faster signals can be observed.
In contrast to passive side channel attacks, fault injection attacks actively influence the operating environment such as the voltage or the system clock of the chip. In Section 4.2, we describe clock and voltage glitching attacks. The idea of these attacks is to induce momentary faults during

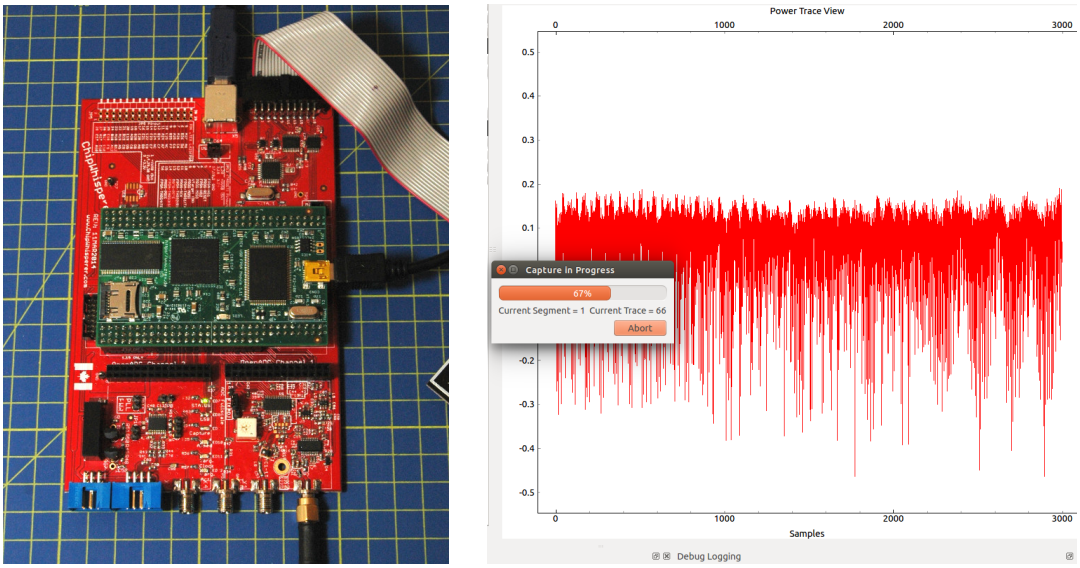

Figure 1.9: FPGA Based DPA Measurement Setup

security critical operations of the chip allowing the attacker to circumvent typical protection mechanisms. For instance, an embedded smart grid device might contain a microcontroller with security critical firmware residing in the chip. The analyst would like to read out the firmware over the microcontroller's programming port but as a countermeasure the device manufacturer has configured the controller's security fuse to prevent firmware readout. If the internal implementation of the microcontroller chip checks the state of the security fuse setting in software, the analyst could resort to a clock glitching attack. During clock glitching one or more very short clock pulses are inserted at the time of the security fuse check. Since the clock speed of these momentary pulses is too high for the controller, some of its more complex internal combinatoric logic blocks will not be finished when the next clock signal arrives whereas lower complexity logic blocks such as the program counter will work as expected. From the context of the software security fuse check, a successful attack would thus cause the program counter to increase while the security fuse comparison operations (i.e., typically a branch instruction) would not finish. As a result, the attacker could thus practically *skip* the security fuse check and read out the firmware even though the security fuse would normally prevent this action. Voltage glitching attacks can lead to a similar result but they utilize a different approach. Depending on the employed memory technologies, the chip's internal memory cells often work by comparing a stored charge (i.e., the state of a stored bit) with a threshold reference voltage. If the attacker can momentarily change the supply voltage of the chip, the threshold reference voltage is changed as well and the read out bit state of the memory cell might no longer represent the stored bit state. Depending on the voltage glitch, an attacker could thus force the chip's internal logic implementation to read out the state of the security fuse in a wrong way allowing him to dump the device's firmware even though this function has been originally disabled through the security fuse setting.

If the chip is not susceptible to non-invasive attacks, an attacker or analyst can resort to semi-invasive and invasive attacks that require the decapsulation of the chip package so that the die is exposed. In Section 4.2, we present our decapsulation process that is based on milling a small cavity into the chip package followed by wet chemical etching with fuming nitric acid, rinsing in acetone and ultrasonic cleaning (Figure 1.10).

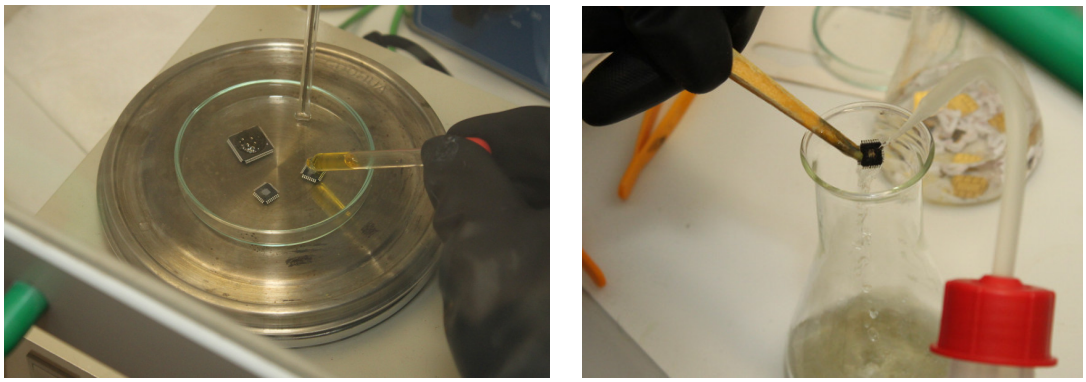

Figure 1.10: Chip Decapsulation with Fuming Nitric Acid (left), Rinsing with Acetone (right)

Depending on the level of invasiveness, significantly more powerful attacks become available. For instance, depending on the utilized memory technology, the content of ROMs might be optically read out. Localized fault injection attacks become feasible by utilizing lasers on active chip regions to cause transistors to switch through even though there is no voltage supplied to their gate. Utilizing a localized laser fault injection attack, an attack could thus circumvent the readout logic of the security fuse to enable firmware readout despite the security fuse setting. To discover the physical location of the security fuse and its readout and buffering logic, the use of IC reverse engineering techniques is common. For IC reverse engineering, the IC is imaged under an optical microscope or a Scanning Electron Microscope (SEM). The reverse engineer analyzes the taken images to gather information on the internal chip structure and functions such as the security fuse or hidden test modes that could be utilized to carry out attacks (Figure 1.11). To obtain the information of all chip layers, it is necessary to carefully remove and image all chip layers. As a result, the attack is no longer semi-invasive but fully invasive. Considering invasive attacks, once again significantly more powerful attacks become feasible essentially bringing PCB level attacks such as probing or signal injection down to the chip level. We present basic protection mechanisms against these kinds of attacks and conclude that critical smart grid devices should rely on secure hardware components instead so that the presented attacks can be mitigated or even prevented in the future.

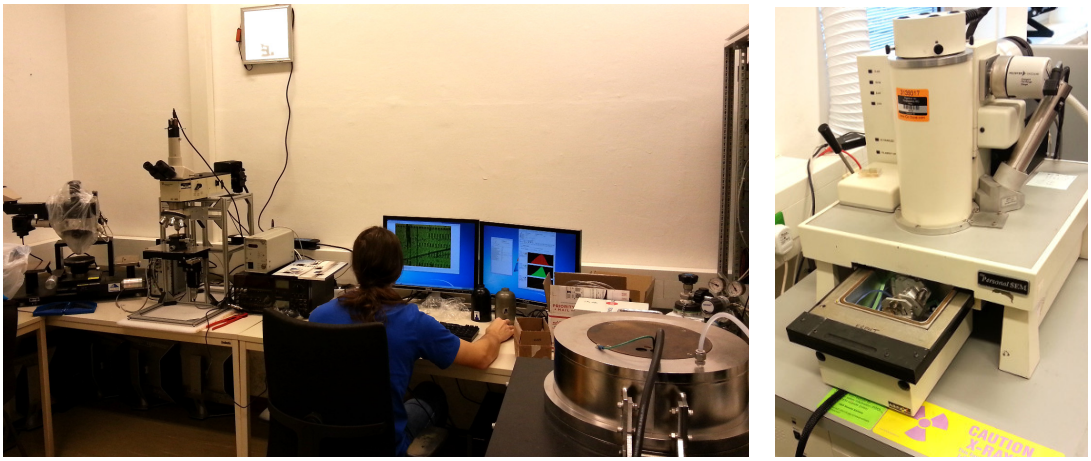

Figure 1.11: Microchip Image Analysis (left), Scanning Electron Microscope (right)

In Chapter 5, we further explore the use of IC silicon reverse engineering techniques to analyze proprietary chip test modes for firmware extraction which has been published under the title "Breaking Integrated Circuit Device Security through Test Mode Silicon Reverse Engineering" at the *21st ACM Conference on Computer and Communications Security (ACM CCS)* in 2014 [54]. During the manufacturing process of Integrated Circuits, a wide range of techniques such as lithography, deposition and etching processes are used to transfer the design to the silicon substrate wafer [125]. Each of those processes can lead to subtle manufacturing defects that would render individual dies on the wafer unusable. As a result, designers typically use "design for testability" approaches by adding test modes to the silicon design. During the manufacturing process these test modes can be utilized to detect and remove faulty dies. In the finished product, more advanced test modes like JTAG can also be utilized for additional purposes such as programming and debugging. In Section 5.2, we describe test modes commonly found in modern IC devices. Common test modes are scan chains, JTAG test functions, Built-In Self-Test (BIST) or proprietary test modes [90]. Scan chains are typically inserted by dedicated scan chain insertion tools. The basic idea of a scan chain is visible in Figure 1.12. Synchronous logic can be described at the Register Transfer Level (RTL) comprising clock driven registers and combinational logic blocks. Considering a single input bit, a combinational logic block is thus sourced from a flip-flop and after some combinational logic delay the result will be available at an output flip-flop. If a scan chain is inserted, relevant flip-flops are extended with an additional multiplexer and a test mode enable input (`shift_en`). If the test mode is not enabled, the flip-flops will behave like ordinary flip-flops. However, if enabled, all scan flip flops are chained together so that the output of one scan flip-flop is used as the input of the next scan flip-flop and a very large shift register is formed. This way during testing, a test vector can be written into the large shift register. Afterwards the device is clocked a few times with the test mode disabled and once re-enabled, the state of all scan flip-flops can be read out and compared with the expected result. This way errors can be efficiently detected and localized. In the finalized IC, the scan chains are typically still available but kept secret. If an attacker manages to uncover the secret mode of operation, arbitrary states can be injected and security critical internal device states could be read out.

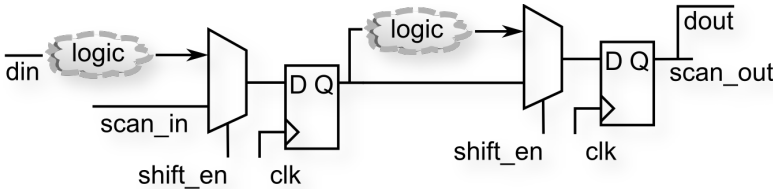

Figure 1.12: Simplified Scan Chain

JTAG test modes are another common way to test ICs during manufacturing. The general idea of JTAG is to implement a state machine (Figure 1.13) that can be utilized over dedicated device pins to specify the test mode, its parameters and to read out the result of the test. In addition to documented test modes that can be utilized by device users, device manufacturers often implement their own secret test modes to verify the function of the die during the manufacturing process. If discovered by attackers, it is often possible to circumvent device security leading to the full disclosure of internal device states. For instance, Skorobogatov et al. recently applied side channel attacks to identify a test mode usable as backdoor inside a high-security chip [100].

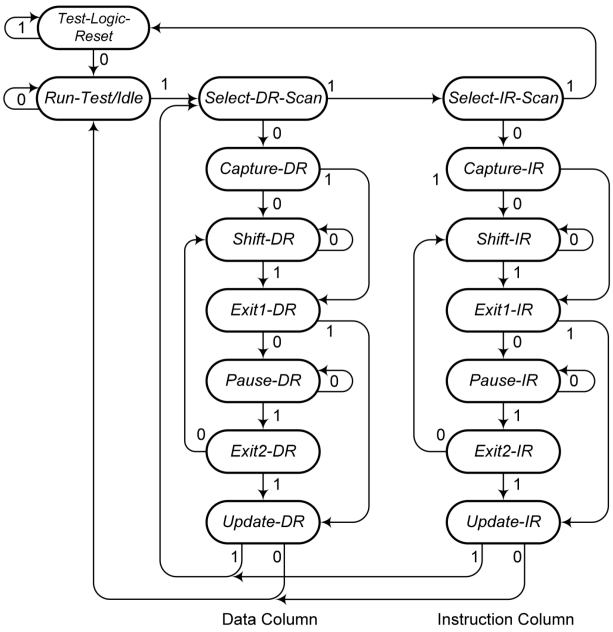

Figure 1.13: JTAG State Machine [5]

Built-In-Self-Test (BIST) is another common method enabling the IC to test itself during startup. The general idea is to generate a pseudo-random test input from a known static seed value. The test input is supplied to the on-chip components and the output is compared with a table containing known good results. Depending on the implementation of these tests, an attacker might be able to abuse the BIST logic to obtain security critical internal device information. Although scan chains, JTAG and BIST are de-facto industry standards, manufacturers can still choose to implement

their very own proprietary test modes. From an attacker's view, utilizing non-invasive techniques such as sniffing, signal injection or side-channel attacks is not promising to determine how these proprietary test modes work and deep silicon analysis is clearly the better choice. In Section 5.3, we present a case study showing how limited effort silicon reverse engineering can be used to uncover the secrets of hidden test modes to extract device firmware. After an initial PCB and signal analysis using conventional non-invasive physical attack techniques such as probing (see Chapter 4 on physical attacks), we decapsulate the chip in concentrated sulphuric acid at a temperature of 170 °C in multiple wet chemical etch rounds (Figure 1.14). After that the chip is cleaned in acetone and isopropanol in an ultrasonic cleaner. To get an overview of the blocks on the die, we took 19x27 (i.e. 513) images on a motorized optical microscope and stitched them together to obtain a detailed 87 Megapixel image. The image analysis shows that the chip has two ROM memories (blocks marked in red with letters *B* and *E*), a RAM (letter *C*), a CPU (letter *D*) as well as an unknown logic block (*A*). In addition, we added the pinout information to the image so that both unknown and known or measured pin functions such as power supply, clock and data pins are visible. The pin functionality is also roughly visible by analyzing the silicon layout. For instance, power supply pins do not have I/O buffers and can be clearly distinguished from data I/O pins. Good clues to identify test pins are thus I/O pins that are unconnected or driven with static voltage levels on the PCB. Since the major building blocks on the IC have been identified, probably locations of test modes are the unknown logic block as well as the CPU block.

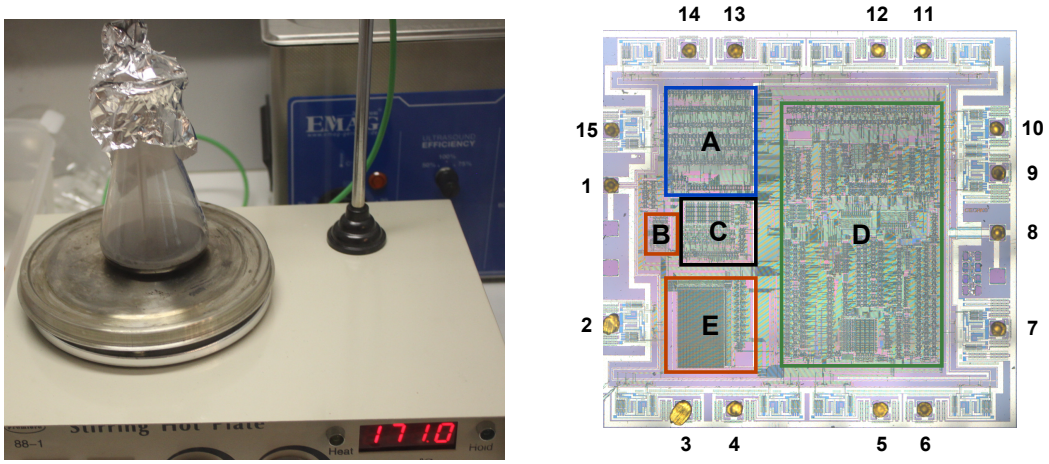

Figure 1.14: Decapsulation in Concentrated H_2SO_4 (left), Commented CIC Die with Manufacturer Chip Label `CECRN8` (right)

For detailed test mode reverse engineering, we imaged interesting IC areas with a Scanning Electron Microscope and stitched the images together. In the detailed analysis, interesting I/O pins that could potentially lead to test mode functionality are used as starting point. From these pins, we followed the signal traces to get clues on the connected logic implementation. Figure 1.15 shows exemplary logic blocks connected to the suspected test mode pins `6` and `7`. The analysis shows that the pins control test mode multiplexers. Further silicon analysis showed that signals from these multiplexers end up in flip-flops (i.e., registers) and logic blocks next to

the ROM. Our investigation finally revealed that there are multiple test modes and that data can be loaded into the instruction register over the external test pins. Within our proof of concept implementation, we were thus able to achieve *arbitrary code execution*.

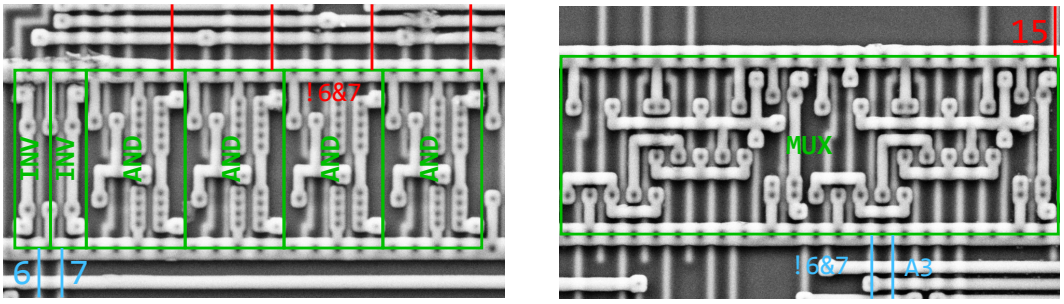

Figure 1.15: Proprietary Test Mode Logic (left), Test Mode Multiplexer (right)

Whenever necessary, we had to delayer the die so that deeper microchip layers become visible during the analysis. For that process, we utilized a lab polishing machine with a 0.1μ hard polishing disc and water as a lubricant (Figure 1.16). The more unwanted material we removed, the shorter our polishing runs followed by optical microscopic analysis became until we finally achieved a satisfying result.

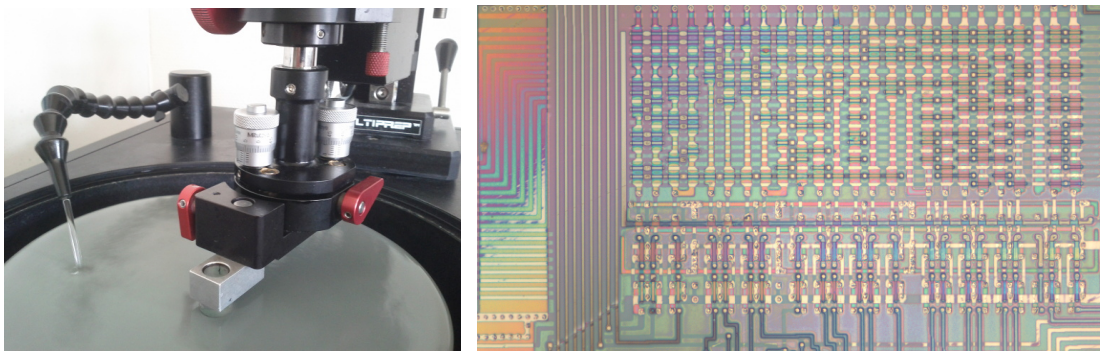

Figure 1.16: Polishing Machine with 0.1μ Polishing Disc and Water as Lubricant (left), Poly Layers below M1 Metal Layer of CPU Instruction Decoder Unit (right)

With the obtained information gained through limited effort silicon reverse engineering, we were now able to fully understand the proprietary test mode in the chip. Through the test mode and the achieved *arbitrary code execution* we were able to inject our own code and to successfully read out the secret code in both of the ROMs. The details of our attack are described in Section 5.3. We disassembled and fully reverse engineered the code from the ROMs in a disassembler. To verify that our findings were indeed correct, we implemented the reverse engineered algorithm on an FPGA and exchanged the real chip on the PCB with our programmed FPGA. Our evaluation shows that the system works as expected and the results we were able to obtain from the extracted code are indeed correct. Our work effectively demonstrates that limited effort silicon analysis is a viable option to discover and reverse engineer secret IC test modes even if they are proprietary

and do no follow de-facto standards. By utilizing these test modes, device security can often be broken and the firmware can be extracted and analyzed as presented.

Embedded Security Audit

As soon as the firmware is available to the analyst, a firmware security audit can be conducted by utilizing vulnerability discovery techniques typically based on static analysis or more powerful dynamic analysis approaches. Unfortunately, mainly due to custom proprietary hardware, undocumented peripherals and strict system limitations, recent publications have shown that in comparison to PC based commodity systems the prevalent vulnerability discovery techniques on embedded systems are still mostly based on static analysis [13, 59, 118]. In Chapter 6, which has been published under the title "PROSPECT: peripheral proxying supported embedded code testing" at the *9th ACM Symposium on Information, Computer and Communications Security (ASIACCS)* in 2014 [56], we describe the typical dynamic analysis challenges on embedded systems and present PROSPECT, a system that enables dynamic firmware analysis by forwarding peripheral device communication from a virtualized analysis environment to the embedded system under test. We outline common vulnerability discovery techniques for binary code on PC based systems and point out that for widely established approaches such as fuzz testing (Section 6.2), dynamic taint analysis or symbolic and concolic execution [17, 94], dynamic analysis is often a key requirement. Considering that a PC system comprises of hardware, an operating system and software applications, a security analyst can dynamically instrument any of these layers by leveraging tools such as state-of-the-art-debuggers, OS centric analysis frameworks such as CWSandbox [127] or Virtual Machine Introspection (VMI) [38]. Although a typical embedded system (Figure 1.17) has an apparent resemblance to a PC system, dynamic analysis is much more challenging.

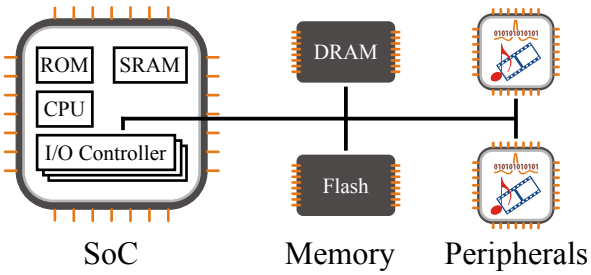

Figure 1.17: A Typical Medium to Large Scale Embedded System

Driven by small size, low power consumption and low prices, embedded systems are often highly resource constrained, access to the file system is limited, and the OS kernel and tools available on the device include just a minimal set of functions to fulfill the necessary tasks of the embedded device product [88]. Under this perspective, many common vulnerability discovery techniques on PC based systems are hard to apply to their embedded systems:

1. Utilizing a debugger to instrument a program is only feasible if the embedded OS kernel includes support for the necessary system calls (i.e., `ptrace()`). However, even with basic kernel support, running a state-of-the-art debugger on the system itself might not be an option to its resource constraints.

2. Instrumenting the operating system typically requires the modification of the kernel or the loading of driver modules. However, since embedded systems typically run customized operating kernels tailored to their specific hardware devices, instrumenting the operating systems might not be feasible as well.

3. Instrumenting the hardware would require virtualization of the entire embedded system including its often proprietary peripheral devices. These proprietary devices are accessed by the firmware. In case a required peripheral device is not available or does not behave like expected, the firmware can no longer by analyzed. With tremendous effort proprietary peripheral devices and their communication interface would thus need to be reverse engineered and implemented as emulation code before the actual security audit could commence. As a result, the full hardware virtualization approach might not be feasible for security testing just the same.

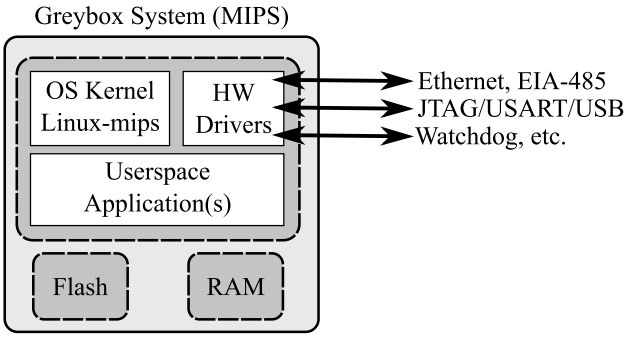

Figure 1.18: A Typical Graybox System Example

Figure 1.18 shows a typical graybox system from the security analysts point of view. For instance, the analyst's goal could be to test the security of a networked userspace application with fuzz testing and dynamic analysis for monitoring. However, for the reasons mentioned above, the analyst needs to face the challenges portrayed above and it is likely that dynamic analysis approaches won't be available. The system thus lacks the support and resources to run security analysis tools on the embedded device itself and the userspace application can not be analyzed in a virtualized analysis environment since the system's peripheral devices won't be available in

the virtual machine. However, one key observation is that the userspace applications within the firmware typically use standardized interfaces such as character devices to communicate with the device driver which in turn interacts with the peripheral hardware. At the system call level, the basic idea of PROSPECT is to transparently forward peripheral device communication from within the virtualized analysis environment to the embedded system hardware. The firmware can thus be executed from within a virtual machine, any device accesses to selected peripheral devices will be forwarded and the firmware can interact with the real hardware. PROSPECT thus allows to overcome many of the dynamic security testing challenges that typically need to be faced on embedded systems, today.
Targeting the Linux platform, we analyzed the source code of the character device drivers in three different Linux kernel versions (Linux-2.4.20, Linux-2.6.38.1, Linux-3.4.4) to identify common system calls that are used to interact with the drivers. Due to the required structure of Linux device drivers, each driver needs to define a `file_operations` field that includes the supported system calls and the handling functions in the driver implementation. Table 1.2 provides an overview of the analyzed number of device driver files and the percentage of files that actually define `file_operations`. We believe that on newer kernels a higher level of device driver abstraction is utilized which is why the amount of files that directly define `file_operations` is lower in comparison to older kernel versions.

Linux-2.4.20			Linux-2.6.38.1			Linux-3.4.4		
files	fops	fops %	files	fops	fops %	files	fops	fops %
264	77	29.17	143	62	43.36	107	54	50.47

Table 1.2: Analyzed Device Drivers on Different Linux Kernel Versions

For those files that contained a `file_operations` field, we analyzed their supported system calls. The result is visible in Table 1.3. For instance, it can be seen that in Linux-2.4.20, 83.12% of the device drivers support the `open` system call while the percentage is a bit lower (74.19% and 77.78%) for Linux-2.6.38.1 and Linux-3.4.4.
Although PROSPECT could forward any of these system calls to the remote system, it is necessary to execute some of them locally while other ones need to be forwarded. For instance, system calls such as `flush`, `sync`, `fasync` or `aio_fsync` can be executed locally if the access to the remote devices is kept synchronized. Other system calls that directly exchange data with the device (e.g., `ioctl`, `read`, `splice_read`, `write` and `splice_write`) need to be executed remotely. The `open` system call is special since both the file descriptors on the local system and on the remote system need to be considered. PROSPECT can handle the listed basic file operations (i.e., system calls) visible in Table 1.4. The table shows the supported system calls and whether they are executed locally or remotely. Our system thus supports all operations that are frequently used for character devices with the exception of `mmap` which has not been implemented due to lacking support in the underlying FUSE (Filesystem in Userspace) framework.

Syscall	Linux-2.4.20	Linux-2.6.38.1	Linux-3.4.4
aio_fsync	-	0.00	0.00
aio_read	-	1.61	1.85
aio_write	-	1.61	1.85
check_flags	-	0.00	0.00
compat_ioctl	-	6.45	7.41
fallocate	-	0.00	0.00
fasync	28.57	11.29	12.96
flock	-	0.00	0.00
flush	14.29	-	-
fsync	0.00	3.23	3.70
get_unmapped_area	2.60	1.61	1.85
ioctl	84.42	-	-
llseek	6.49	32.26	29.63
lock	0.00	0.00	0.00
mmap	18.18	12.90	14.81
open	83.12	74.19	77.78
poll	32.47	20.97	25.93
read	68.83	82.26	85.19
readdir	0.00	0.00	0.00
readv	0.00	-	-
release	77.92	62.90	66.67
sendpage	0.00	0.00	0.00
setlease	-	0.00	0.00
splice_read	-	0.00	0.00
splice_write	-	0.00	0.00
unlocked_ioctl	-	51.61	50.00
write	62.34	50.00	55.56
writev	0.00	-	-

Table 1.3: Usage of Linux `file_operations` in Character Device Drivers (Percentage)

```
#define _IOC(dir,type,nr,size)    \
((dir << _IOC_DIRSHIFT)|(type << _IOC_TYPESHIFT)|\
 (nr   << _IOC_NRSHIFT) |(size << _IOC_SIZESHIFT))
```

Listing 1.1: Encoding for well-formed IOCTLs

The IOCTL (I/O control) mechanism supports two types of IOCTLs: Well-formed and unrestrictive IOCTLs. Well-formed IOCTLs use the encoding visible in Listing 1.1 so PROSPECT can determine whether the call is supposed to read from or write to the remote process space. However, unrestrictive IOCTLs do not have encodings and, as a result, PROSPECT can not determine the direction of the transfer, how much data is supposed to be transferred and whether provided parameters are supposed to be pointers. We addressed this challenge by introducing a technique, we denote as *dynamic memory tunneling* which is described in detail in Section 6.4. The general idea of the concept is to use a heuristic to determine whether a parameter could be

Syscall	supported	implemented through	local/ remote
aio_fsync	yes	fsync	local
aio_read	yes	read	remote
aio_write	yes	write	remote
check_flags	no	-	-
compat_ioctl	yes	ioctl	remote
fallocate	no	-	-
fasync	yes	fsync	local
flock	no	-	-
flush	yes	fsync	local
fsync	yes	fsync	local
get_unmapped_area	no	-	-
ioctl	yes	ioctl	-
llseek	yes	llseek	remote
lock	no	-	-
mmap	no	-	-
open	yes	open	remote
poll	yes	poll	remote
read	yes	read	remote
readdir	no	-	-
readv	no	-	-
release	yes	release	remote
sendpage	no	-	-
setlease	no	-	-
splice_read	yes	read	remote
splice_write	yes	write	remote
unlocked_ioctl	yes	ioctl	remote
write	yes	write	remote
writev	no	-	-

Table 1.4: Basic `file_operations` supported by PROSPECT

a pointer within a mapped memory region (pointer detection). If the parameter is a potential pointer, the memory region is accessed and a multiple of the kernel's page size is transferred to the target device. Here, the `ioctl` call is executed and the resulting buffer is compared with the transferred buffer. At this point, the difference between those buffers as well as the return value of the `ioctl` call is returned to the originating system.

Figure 1.19 provides an overview of the PROSPECT implementation. On the left side, the virtualized analysis environment based on QEMU is visible. Within the system, parts of the firmware under analysis (i.e., userspace applications) can be instrumented with a debugger or more advanced dynamic analysis techniques. Since the analysis environment has its own kernel and the emulator uses the resources of the PC based host system, the security analyst is typically neither tool nor resource constrained. Whenever the application under analysis needs to access peripheral hardware devices, it will utilize character devices to communicate with the

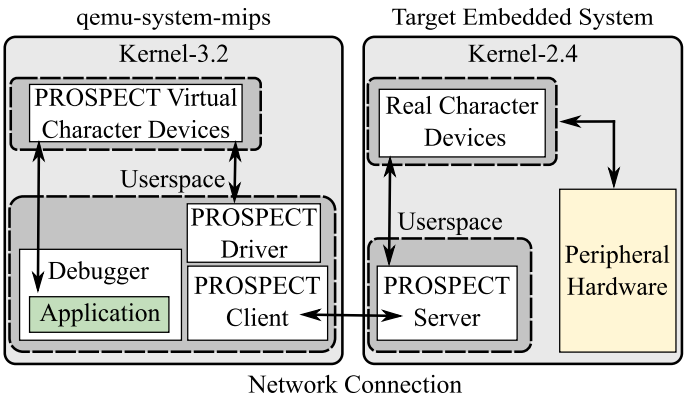

Figure 1.19: Peripheral Character Device Forwarding

driver. The driver would it turn use hardware interfaces to interact with the peripheral hardware device. However, under the analysis environment neither the potentially proprietary driver nor the embedded system peripheral hardware are available and the application would typically fail to execute. PROSPECT addresses this challenge with virtual character devices and transparent forwarding to the target embedded system visible on the right side of Figure 1.19. Within the analysis environment, our system comprises a PROSPECT client and userspace driver. With the help of the userspace driver, each client instance will create a unique virtual character device that can be accessed by the application under analysis. Depending on whether system calls should be executed locally or on the remote side (Table 1.4), the calls are forwarded to the PROSPECT server stub on the target embedded system. At this point, the system call can be executed on the real character device to interact with the attached peripheral hardware. The results are returned to the PROSPECT client and are finally returned to the application under analysis through the virtual character device. Since the server stub needs to be executed on the resource constrained target system, it comprises a lightweight implementation with minimal system and software requirements. The full description of our implementation is available in Section 6.4 including details on how we addressed concurrent device accesses, file descriptor tracking or IOCTLs with our dynamic memory tunneling technique. In Section 6.5, we evaluated PROSPECT in two ways by measuring its performance impact based on system call timing and by conducting a full-scale security audit on a real-world embedded firmware alarm system over a period of more than 6 months as a case study. For the performance evaluation, we selected system calls typically used for character device accesses (Table 1.5) and executed them within the analysis environment with PROSPECT as well as on a native 324 MHz embedded Linux MIPS system with 16MiB RAM. We utilized the `strace` tool to collect timing information on $196,075$ system calls on the analysis environment and on $166,972$ system calls on the native embedded system. Our results are visible in Table 1.6 and show that especially the `open` and `ioctl` system calls cause a significant slowdown. The `open` performance impact is caused by the connection establishment between the PROSPECT client and server on the remote system while the `ioctl` slowdown stems from the dynamic memory tunneling mechanism described in Section 6.4. At the same time, our results indicate for typically heavily used system calls such as `lseek()`,

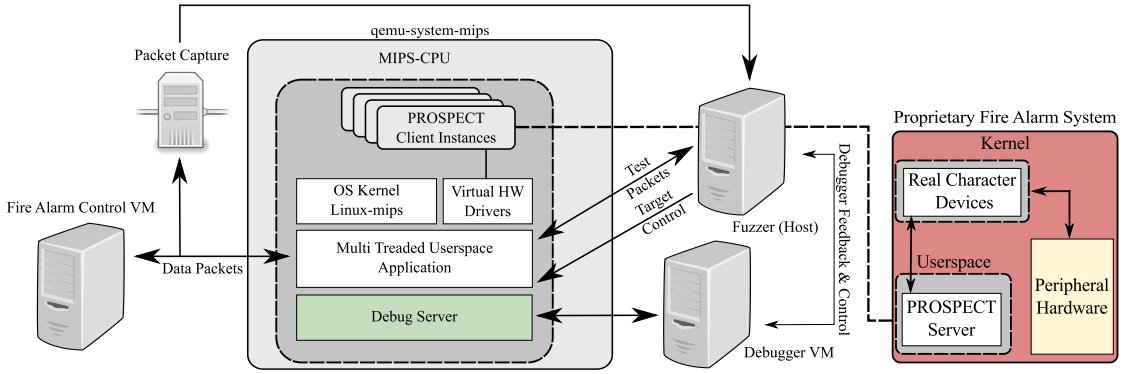

Figure 1.20: Security Analysis Environment

`read()`, `write()` and `_newselect()`), the performance impact is practically insignificant considering that the main use of PROSPECT is enabling dynamic analysis techniques such as program debugging (i.e. single stepping) and code analysis. A full description of our performance evaluation is available in Section 6.6.

In our case study in Section 6.5 involving a practical security audit of a commercial embedded fire alarm system, we utilized PROSPECT in the security audit setup visible in Figure 1.20 to forward more than $500,000$ system calls per analysis run. Overall, the firmware under analysis used 29 multi-threaded processes that concurrently access 5 different peripheral devices which have all been forwarded from our analysis environment to the embedded target system. Within our setup, we performed extensive fuzz testing with dynamic analysis for monitoring as well as dynamic code analysis such as debugging and manual single-stepping through the code. Our security tests revealed a previously unknown zero-day vulnerability in the system and showed that PROSPECT can be utilized for practical real-world application. Nonetheless, PROSPECT has limitations on its own including the slowdown potentially preventing the analysis of time-critical real-time systems, the current lack of `mmap` support due to FUSE limitations or the server's requirement of `pthread` support on the target system (Section 6.7).

Operation	Function
`close()`	Close device
`ioctl()`	I/O Control mechanism
`lseek()`	Seek to a given position
`_newselect()`	System call used for `poll()`
`open()`	Open device
`read()`	Read data from device
`write()`	Write data to device

Table 1.5: System Calls used for Character Device Access

Syscall	Native [%]	Native [ms]	Fwd. [%]	Fwd. [ms]	Diff. [ms]	Slowdown [x]
`write()`	21.27	6.07	22.79	25.39	19.32	3.18
`lseek()`	28.14	0.12	18.88	2.31	2.2	18.65
`ioctl()`	1.6	0.89	4.27	117.15	116.26	130.37
`_newselect()`	14.8	43.27	17.14	40.85	-2.41	-0.06
`read()`	34.16	1.03	36.9	3.29	2.26	2.19
`close()`	0.01	0.1	0.0	N/A	N/A	N/A
`poll()`	0.0	N/A	0.0	N/A	N/A	N/A
`open()`	0.02	0.9	0.02	684.62	683.72	757.37

Table 1.6: PROSPECT Slowdown

In Chapter 7, which has been published under the title "Embedded Security Testing with Peripheral Device Caching and Runtime Program State Approximation" at the *10th International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)* in 2016 [50], we extend the idea of PROSPECT by introducing an intermediary cache. The basic idea is that many vulnerability discovery techniques such as fuzz testing are highly repetitive. For instance, a security analyst might specifically focus on a small security critical firmware code region (e.g., communication handling routines executed prior to user authentication) to discover vulnerabilities exploitable by unauthenticated attackers. If techniques such as fuzz testing are applied, many highly similar test cases will be handled by the firmware under analysis. Since mostly the same code fragments will get executed over and over again during these tests, also the peripheral device interaction within those code fragments will be highly similar. Considering a transparent peripheral device communication proxy such as Avatar [134] or PROSPECT [56] (Chapter 6), adding a caching mechanism (Figure 1.21) could render existing dynamic analysis approaches more powerful by allowing techniques such as snapshotting, test parallelization or testing without physical access to the embedded system. The challenge is not the cache itself, but the caching strategy to decide whether a specific device response is already in the cache and which response should be returned to the firmware at which time. Whenever the cache receives a device request from the proxying framework, it needs to decide whether it is a cache *hit* or *miss*. In case of a cache hit, the device response is already in the cache and it can be returned to the framework without interaction with the real embedded hardware. Otherwise, in case of a cache miss, depending on the expected device state it might be necessary to bring the hardware into the necessary state as well before the actual request can be executed. This can be done by resetting the hardware and replaying prior device interactions the program under analysis performed until this point. The cache thus needs to retain the device request and response history as well. In Section 7.3, we implement and practically evaluate three different peripheral device communication caching strategies: (1) choosing responses by command, (2) choosing responses by command and command history as well as (3) choosing responses by program state approximation.

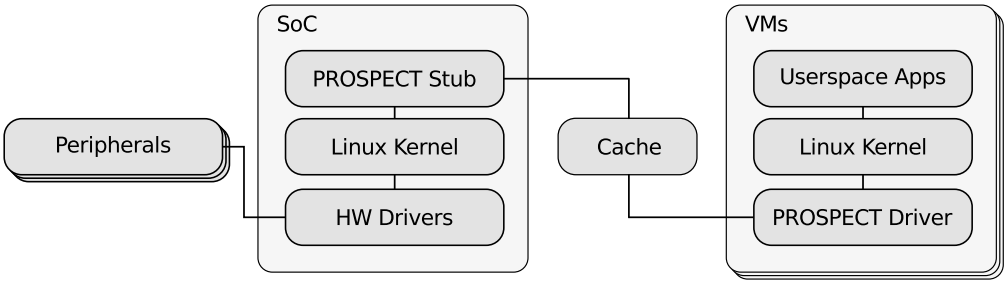

Figure 1.21: Embedded System Testing Utilizing PROSPECT with an Intermediate Cache.

Choosing Responses by Command In case of stateless or very simple stateful peripheral devices such as a controllable switch, the cache does not need any additional information than the device request command itself. For instance, in case of a switch, the `open` and `close` commands are sufficient for the cache. In case of an `open` command, it would just return that the device has been turned on while for the `close` command it would confirm that the device has been turned off. Since the firmware typically depends on successful device operations, it would continue its normal execution in presence of the cache. However, as soon as a device command can have multiple return values, the approach can no longer be utilized. An example would be a realtime clock (RTC) module that would return a new timestamp each time the device is read.

Choosing Responses by Command and Command History An improved strategy is to consider the communication history with the peripheral device as well. For instance, in case of a `toggle_switch` command, the command would either turn on or turn off the switch depending on its current state. If the command is supposed to return the current state of the switch, the cache needs to consider previously issued commands to know the current state of the peripheral hardware device to return the expected response to the firmware. Another example would be a counter peripheral that returns a new counter value each time the device is read. The example of the realtime clock (RTC) module can be seen as a special case of a hardware counter. During a first *training execution* the cache could thus learn from the previous interaction with a device so that the expected replies can be replayed during later firmware executions. However, even if the peripheral device interaction during similar firmware execution runs remains deterministic, the approach fails as soon as the same device is accessed by multiple executions threads at the same time. In this case, even for the same program input the thread scheduler will cause a different execution order of the threads. From the perspective of the cache, the program will thus no longer be deterministic and it becomes unclear which device response is expected by the firmware. In Section 7.3, we provide an example program that illustrates the problem.

Choosing Responses by Program State Approximation A more advanced strategy is to make the cache aware of firmware program state. Whenever a program is executed, it will make use of resources such as the CPU and stack memory. The CPU state is determined through its registers (i.e., registers such as the instruction register, the stack pointer or the CPU's general purpose registers). However, considering typical program constructs such as loops of function calls, it is

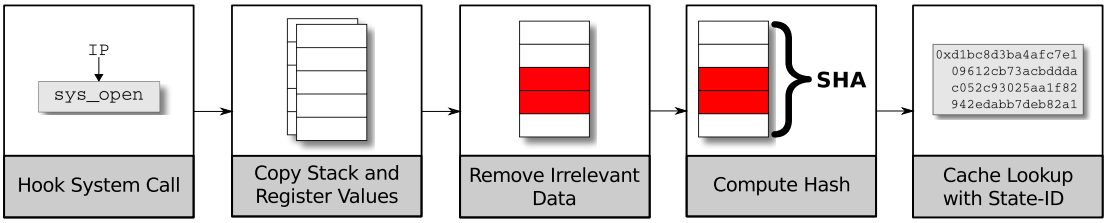

Figure 1.22: State Approximation Heuristic.

likely that content of the register set will be highly identical for different loop iterations. Since the stack memory is used to store the local function frames it needs to be considered by the cache as well. Determining the exact state of a program is a known to be hard problem related to program slicing [21,61]. In contrast, for the cache it would be sufficient to determine an approximation of the program state. The approximation heuristic thus needs to be coarse enough to be practically feasible and, at the same time, it should be precise enough for the cache to determine which device response should be returned to the firmware. In the following, we explore the use of program state approximation as peripheral device caching strategy. A more detailed description is available in Section 7.4.

Figure 1.22 shows an overview of our program state approximation heuristic. In the first step and similar to PROSPECT, we hook the system call executed on the peripheral device. Depending on the system call, the peripheral cache can determine the type of device operation that the firmware requested to perform. In the second step, during system call execution, we retrieve the CPU register state and the last fragment of the stack memory from the suspended process. However, not the entire register set and stack memory region will be relevant for program state approximation. If the state approximation level is too fine, many different states will be determined for the same peripheral device interaction and the cache will not be able to return already stored device responses. On the other hand, if the approximation level is too low, different device interaction states would be combined into a single state and the cache would return wrong device responses. To address these issues, in the third step, we perform a manually optimized weeding step to remove irrelevant data. In the fourth step, we use the remaining information to compute a hash digest denoted *State-ID* that is used for peripheral device response lookups in the cache. To conduct our feasibility study, we created two implementations based on (1) Virtual Machine Introspection (VMI) and (2) a kernel module.

Virtual Machine Introspection For VMI, we extended the QEMU emulator to hook system calls in the kernel to obtain low-level state information from physical memory under consideration of relevant kernel internals. While the VMI approach is very powerful due to having access to all system resources below the operating system kernel, the disadvantages are that the functions necessary for parsing internal memory structures are kernel-dependent and the disabling of QEMU's Translation Block Chaining (TBC) required for reliable system call hooking sufficiently reduces the overall system performance.

Kernel Module Our second implementation utilized a loadable kernel module that performs system call hooking from within the kernel. Compared to the VMI approach, the kernel module significantly simplifies the access to swapped out memory regions and kernel structures. However, in contrast to VMI, the program state approximation logic is executed from within the system emulated by QEMU and thus suffers from a performance impact on its own.

While we suggest to address these performance issues with a hybrid approach in future work (Section 7.6), the main goal of our feasibility study was to determine whether peripheral device caching with program state approximation is a viable approach. We evaluated our approach on programs from the well known GNU core utilities and divided them into low, medium and higher complexity programs. Low complexity programs such as `cat`, `head`, `sum` and `wc` do not use dynamic memory management and only a small subset of the registers was sufficient to correctly approximate the program state for peripheral caching. Medium complexity programs such as `expand` strongly rely on dynamic memory management functions and, as a result, our current lack of heap consideration led to duplicate program states. While multiple training executions and minor manual adaptations of the weeding step rules solved the issues, without improvements such as stack unwinding and heap analysis, medium complexity programs currently represent the limit of our approach. Higher complexity programs such as `sort` not only heavily rely on dynamic memory management, but information highly relevant to the program state are stored on the heap as well. As a result, our approach returns a large number of duplicate states and, similar to symbolic execution, suffers from the well known state explosion problem. Although we believe that our approach can be significantly improved with stack unwinding and dynamic memory allocation/pointer tracking, we expect higher complexity programs to remain challenging.

In addition to embedded software security analysis, the security analyst needs efficient methods to test the practical security of the cryptographic implementation and setups on these systems. For instance, a networked embedded system might employ a cryptographic authentication routine to deny unauthorized users the access to its core functions. The practical security of the authentication scheme thus not only relies on the security of the cryptographic design and algorithms, but it also depends on properties such as the security of the implementations of this design, the choice and strength of the key material or the quality of the random numbers. In Chapter 8, we show at the example of the widely utilized WPA2-Personal cryptographic authentication protocol how low-cost FPGA clustering can be used for high-speed brute force attacks. WPA2-Personal is a strong cryptographic authentication protocol relying on established primitives. Without the discovery of cryptographic vulnerabilities in the scheme, the practical security of many (embedded) systems in the field thus relies on the choice of the password. If the attacker can break the password within a reasonable time, the security of the system can be broken even though the utilized cryptographic authentication primitives are secure by design. While in a secure cryptographic system especially random passwords can only be found if all password candidates are tested one by one, the practical feasibility of those brute force attacks is limited by the speed the attacker can test the password candidates. To achieve significantly higher speeds in comparison to CPU or GPU based systems, we present a highly optimized scalable and fully pipelined FPGA based system. Our system uses 36 low-cost FPGAs combined into a cluster to bring the performance of today's highly expensive professional brute-force attack systems to amateurs with a small budget. In comparison to the currently fastest FPGA based

commercial system, we claim that on the same Kintex-7 based hardware our implementation is more than 5 times as fast. To achieve high brute-force attack speeds, it is necessary to analyze the cryptographic WPA2-Personal authentication protocol (i.e., the WPA2-Personal handshake) in full detail.

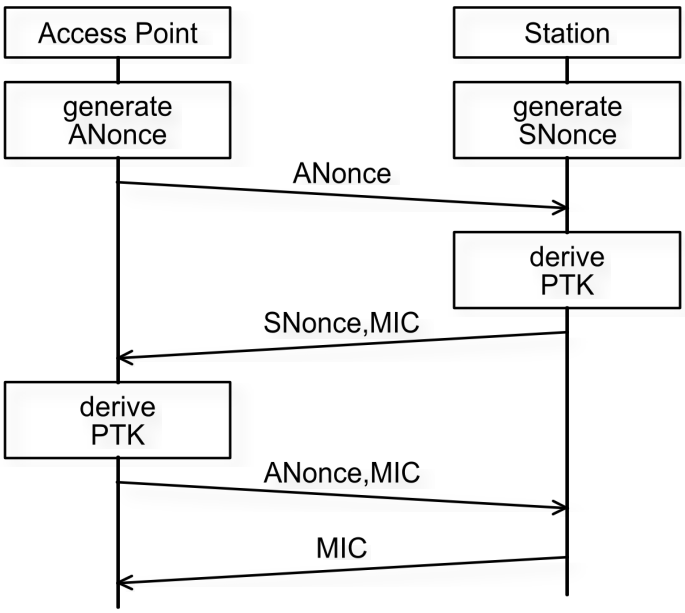

Figure 1.23: WPA2-Personal 4-Way Handshake

The 4-way handshake is visible in Figure 1.23. Initially, both Access Point (AP) and Station generate a 32 byte random nonce value. After the Station has notified the AP to initiate the authentication protocol, the AP starts the 4-way handshake by transmitting its `ANonce` value to the Station. At this point the Station starts to derive the Pairwise Transient Key `PTK` through the WPA2-Personal key derivation function visible in Figure 1.24. During the computation of the `PTK`, the first step involves the computation of the Pairwise Master Key `PMK` with the well known PBKDF2 [49] salted key derivation function. The Wi-Fi password is used as key while the network's SSID (Service Set Identifier) is used as cryptographic salt. At the core of WPA2-Personal's PBKDF2 key derivation scheme is the SHA1 cryptographic hash function in HMAC construction (Figure 1.25). Since SHA1 outputs a 160 bit hash digest, the HMAC output as well as the output of the PBKDF2 function will be 160 bits long as well. Nonetheless, the length of the `PMK` needs to be 265 bits long which is the why two consecutive PBKDF2 rounds need to be performed. Their output is concatenated and truncated to obtain the 256 bit `PMK` from the two 160 bit outputs. Internally, PBKDF2 uses the salt combined with a 32-bit round counter to perturb the key computation so that for the same salt and key inputs the PBKDF2 output will be different for each round.
After the Station has obtained the `PMK`, it computes the `PTK` and its truncated variant `KCK` through the pseudorandom function PRF-128. Internally, the function relies on HMAC-SHA1 as well. The `PMK` is used as key input while network parameters such as the physical addresses of the

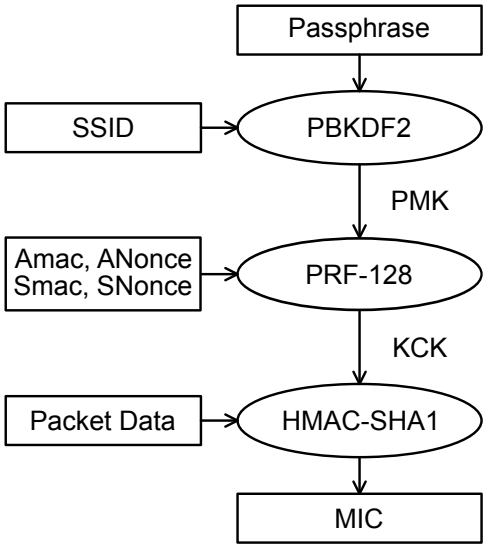

Figure 1.24: WPA2-Personal Key Derivation Function

Access Point (`Amac`) and the Station (`Smac`) as well as their nonce values `ANonce` and `SNonce` are used as salt. At this point in the 4-way handshake, the Station can send its `SNonce` to the Access Point. However, in contrast to the first message in the handshake, the message is digitally signed with a Message Integrity Code (MIC) that is computed from the message and the `KCK` with the help of HMAC-SHA1. The Access Point can thus verify that the Station has knowledge of the secret password and it can compute the key material including the `PTK` just the same. Since the Station has not verified whether the Access Point knows the network password as well, the Access Point returns a signed message with its `ANonce` to the station. If the message signature was authentic, the Station knows that the Access Point has the network password as well. In this case, the 4-way handshake is completed by sending a usually empty but signed message back to the Access Point to confirm successful network authentication. A more detailed description of the WPA2-Personal 4-way handshake and the key derivation can be found in Section 8.2.

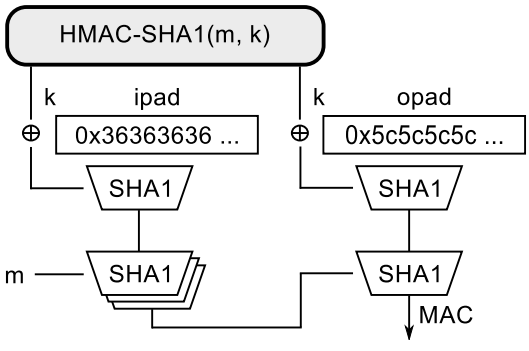

Figure 1.25: PBKDF2 Core with SHA1 Rounds in HMAC Construction

An attacker can capture the 4-way handshake and obtain knowledge of the nonce values `ANonce` and `SNonce` as well as of network data relevant for the key derivation such as the physical addresses of the Access Point and the Station or the SSID. For any of the three signed messages in the handshake, the attacker knows the message content and the message integrity code (MIC). During a password guessing attack, for each password candidate the `KCK` needs to be derived. With the `KCK`, the attacker can compute the MIC over one of the known-plaintext messages and compare the computed MIC code with the MIC code from the message. If the code matches, the correct password has been found.
To protect users from password guessing attacks, the passwords needs to be at least 8 characters long and the described WPA2-Personal key derivation scheme relies on a high number of SHA1 iterations. Each PBKDF2 round has $4,096$ HMAC-SHA1 iterations where each of those requires at least 2 SHA1 iterations. In total, to derive the `KCK` and compute a MIC, there are at least $16,396$ SHA1 iterations necessary for each password candidate (see Section 8.2 for details). To mount a successful attack within a reasonable time, the attacker thus needs to be able to achieve a very high SHA1 computation performance and the Wi-Fi password itself needs to be weak. The higher the computation performance, the more stronger passwords can be attacked within a reasonable time frame. We analyzed the SHA1 internals and found it to be ideally suited for efficient FPGA implementation due to its low memory requirements, the realization of its rotate and shift operations through FPGA interconnects, the low implementation complexity of algebraic functions in FPGA Look Up Tables (LUTs) and the possibility to create a fully pipelined implementation for its internal 80 compression rounds. The most expensive operations within an FPGA implementation are SHA1's additions due to the long carry chain between the adders.

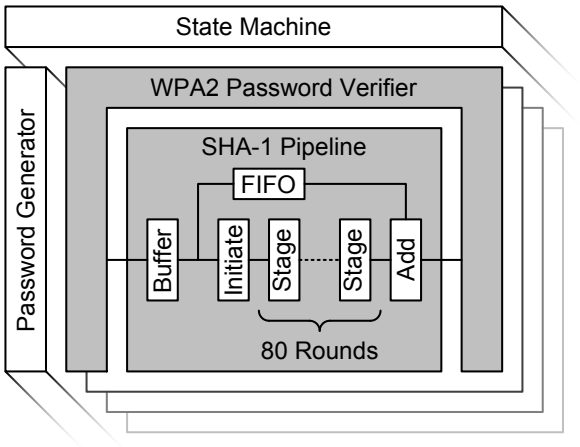

Figure 1.26: FPGA Design Overview

For efficient high-speed brute force attacks on WPA2-Personal, we explore the use of low-cost FPGA boards that have been previously used for cryptocurrency mining. An overview of our design is visible in Figure 1.26. The system consists of a global password (candidate) generator and a global state machine to orchestrate the brute force attack. At the heart of our design are

multiple WPA2 password verifier cores where each one is utilizing a highly optimized 83 stage SHA1 pipeline. As a result, each of the cores computes 83 different password candidates in parallel at maximum FPGA clock speed due to our optimizations. Once the pipeline is filled, we obtain a full SHA1 computation per clock cycle.

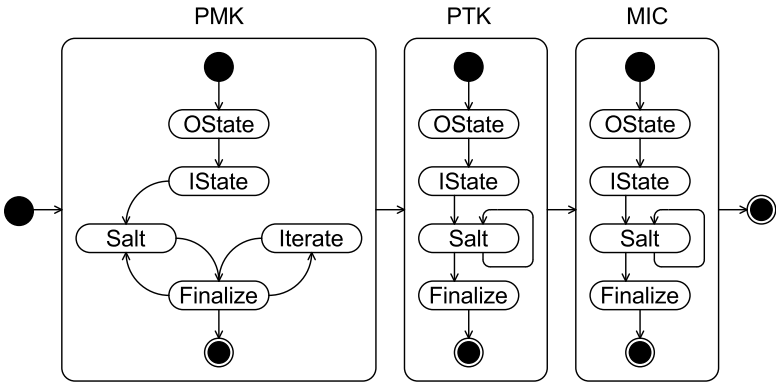

Figure 1.27: WPA2-Personal FPGA States

Figure 1.27 provides an overview of the states of a password verifier core. Initially for each password candidate, the `PMK` needs to be derived with two PBKDF2 key derivation function rounds. Once the `PMK` is available, the `PTK` and its truncated variant `KCK` can be computed. In the last step, we compute the `MIC` code for a known message to determine whether the password candidate is the correct Wi-Fi network password. Since all three derivation phases (i.e., PBKDF2, PRF-128 and the final MIC computation) rely on HMAC-SHA1, the computation steps of the HMAC construction such as outer and inner state computation, internal iterations for salting and hash finalization are similar to each other. A detailed description of the state machine is available in Section 8.3.

To start a brute force attack round, the global state machine uses the password generator (Figure 1.28) to generate password candidates. As long as the `enable` signal is asserted, the password generator will output a new password candidate per clock cycle. Each password verifier core is fed with 83 password candidates until all cores have filled up their pipelines and can start computation. During the computation, the password generator is disabled. Since the cores of the password verifier are filled up sequentially, they will complete their work in the same order. As soon as the first verifier core is finished, the core and the subsequent ones will be checked by the global state machine whether they were able to identify the correct password. If it is found, the password can be read out from the FPGA. Otherwise, the global state machine restarts the password generator and the next brute force attack cycle begins.

The key to high performance significantly lies in the highly optimized design of our SHA1 pipeline. While SHA1 has 80 rounds and a fully pipelined implementation would thus have 80 pipeline stages as well, not all pipeline stages perform the same operations. For instance, in the last round of SHA1 an expensive addition operation needs to be performed. Considering the critical path in FPGA design, the last round computation as well as the finalizing addition would need to be performed in the same clock cycle. As a result of the additional adder logic,

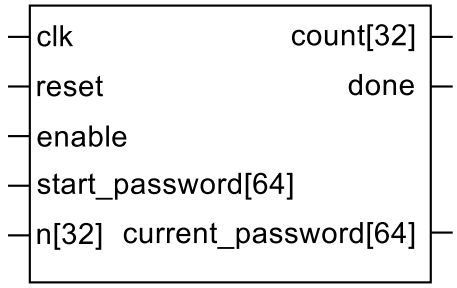

Figure 1.28: Password Generator Block

the maximum clock frequency of the entire design would decrease leading to a lower system performance. Similarly, the logic required to feed the first pipeline instance with data from the global state machine would also have a negative impact on the maximum clock speed. Our SHA1 pipeline thus makes use of 3 additional pipeline stages. The first one is a *buffer* stage to remove the negative performance impact of the feeding logic. Our analysis of the SHA1 algorithm's structure also showed that the 4 expensive finalization additions can be split up. The *initiate* and the finalizing *add* stages perform the finalizing SHA1 additions where the first addition stage is located at the beginning of the pipeline and uses a Block-RAM based delay line to reduce the number of signals that need to be forwarded (and physically routed on the FPGA) throughout the pipeline. Besides the optimizations in our SHA1 pipeline stage structure, we extensively conducted other optimizations such as the improvement of the SHA1 message expansion steps, boosting of FPGA shift register inferences, minimizing the number of necessary FPGA interconnects and bus sizes as well as making use of FPGA floor planning to reduce routing delays. Our optimizations are covered in detail in Section 8.3.
To evaluate the performance of our design, we created implementations for newer model Xilinx Artix-7 FPGAs (Figure 1.29) as well as for older model Xilinx Sparten-6 devices in a cluster (Figure 1.30). The overall system design overview is visible in Figure 1.31. We created a software tool on a host PC that is responsible to continuously send password brute force work packages to the FPGA boards. Depending on the FPGA board, each board includes a EZ-USB FX2 controller and either one (Artix-7) or four (Spartan-6) FPGAs. The firmware on the EZ-USB FX2 controller can be controlled by the software running on the PC via USB. The controller selects the requested FPGA and communicates with it via a bus. The software on the PC also continuously checks for computation errors on the FPGAs caused by too high operating temperatures. If errors are detected, the FPGAs are automatically clocked at a lower clock speed and the erroneous brute force work packages are resubmitted. With the newer Artix-7 devices, we use the internal temperature sensors so that depending on device cooling, the FPGA individually scale the operating frequency to achieve optimum system performance. A more detailed description is available in Section 8.3.

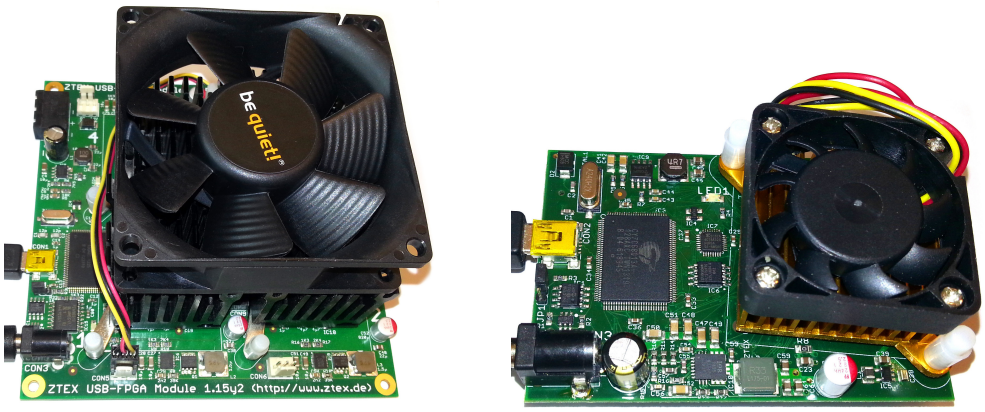

Figure 1.29: Ztex 1.15y Board (left), Ztex 2.16 Board (right)

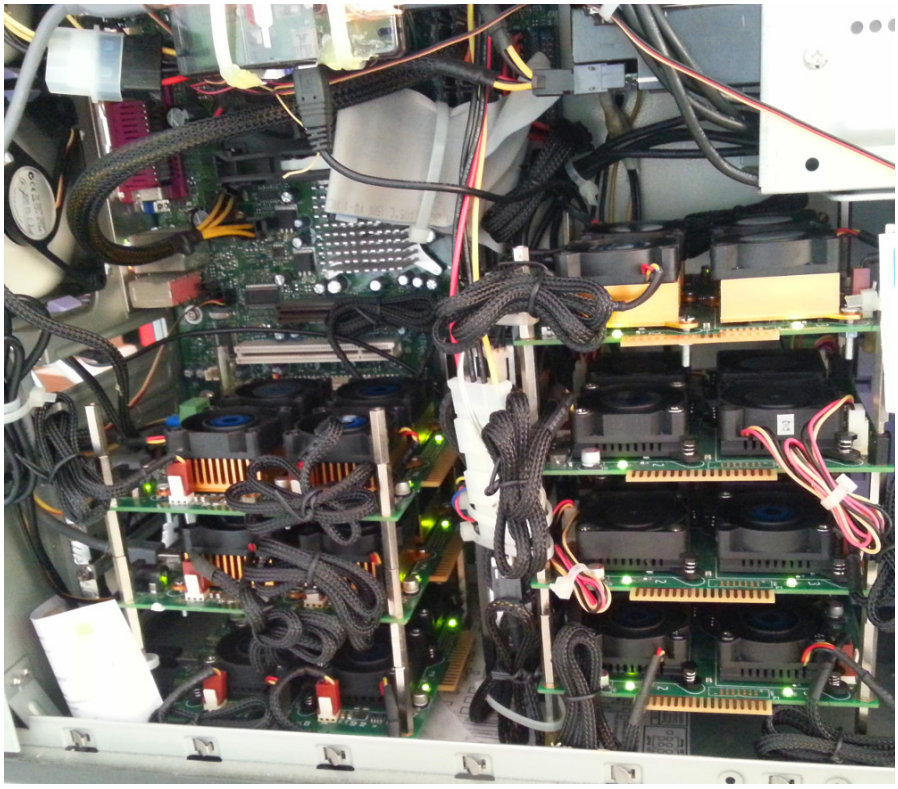

Figure 1.30: Spartan-6 XC6SLX150T Cluster

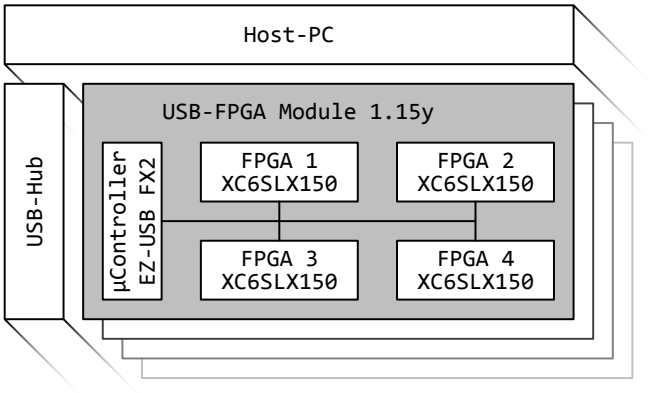

Figure 1.31: System Overview (Spartan 6 System)

We evaluated the performance and power requirements of our system and compared it to GPU based systems as well. The results are visible in Table 1.7 and show that our solution not only brings the performance of highly expensive profession systems to amateurs, but in comparison to GPUs our FPGA implementation also allows significantly higher brute force speeds at a lower power consumption and comparable prices. The details of our FPGA evaluation and the achieved results can be found in Section 8.5.

System	FPGAs	Type	Cost	Cores	Tool W	Tool MHz	Meas. W	Act. MHz	calc pwd/s	pwd/s	pwd/s W
Ztex 1.15y	1	XC6SLX150T-3	175	2	4.281	187	6.99*	180	21,956	21,871	3,128*
Ztex 1.15y	4	XC6SLX150T-3	700	8	17.124	187	27.96	180	87,826	87,461	3,128
9x Ztex 1.15y	36	XC6SLX150T-3	2,400	72	154.116	187	254	180	790,436	741,200	2,918
Ztex 2.16	1	XC7A200T-2	213	8	10.458	180	11.04	180	87,826	87,737	7,947
N/A	1	XC7K410T-3	2,248	16	25.634	216	N/A	N/A	210,783	N/A	N/A
N/A	48	XC7K410T-3	107,904	768	1,230.432	216	N/A	N/A	10,117,584	N/A	N/A

Table 1.7: Performance and Power Results of our Implementations for Different FPGA Devices and Systems/Boards

In addition to the performance evaluation, we also evaluated the impact on practical systems in the field. We conducted a real-world case study involving more than $166,000$ Wi-Fi networks in Austria and its border regions (Figure 1.33). The Wi-Fi networks we analyzed are set up on the customer's cable modems by the largest ISP in our country. Our analysis showed that these modems utilize random default passwords comprising only 8 uppercase character Wi-Fi passwords (Figure 1.32). Our practical evaluation using Wi-Fi test vectors showed that with our Xilinx Spartan-6 based FPGA cluster we can recover the password for each of those networks within no more than 3 days per network. More details on the real-world evaluation and the obtained results can be found in Sections 8.4 and 8.5. Our results thus indicate that embedded systems leveraging cryptographic authentication protocols can not only be tested at high-speeds with low-cost FPGA based clusters, but our real-world case study also highlights the practical impact and the necessity of such security tests.

Figure 1.32: Bottom Side of a Cable Modem

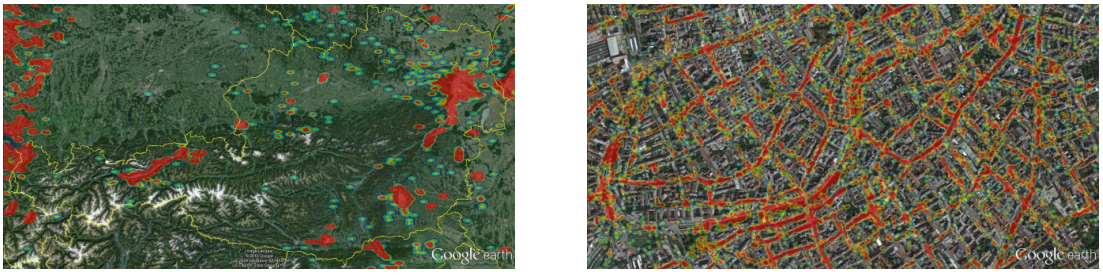

Figure 1.33: Density of `UPC<n>` Networks with Potentially weak WPA2-Personal Passwords

Risk Mitigation

Considering security critical embedded systems such as the ones employed in critical smart grid infrastructures, the last step is to mitigate the identified risks with a subsequent compliance checking step (Figure 1.34). In Section 3.2, we define a methodology to assess the risk potential based on the expected probability and severity of an attack. Identified risks need to be mitigated on an individual basis and under strong consideration of the identified vulnerabilities during the technical risk assessment and security auditing steps. The goal of risk mitigation is thus to either decrease the probability of successful attacks, or to alleviate the potential severity of those attacks to an acceptable level. In the $(SG)^2$ KIRAS project under national FFG grant number 836276, we defined a catalog that provides generic smart grid risk mitigation measures. The catalog builds up upon the architecture model and the risk assessment approach presented in Chapters 2 and 3. For each of the 32 threat clusters, we provide generic measures including best practice examples that can be taken to mitigate the risks. The mitigation strategies have been discussed and evaluated with smart grid experts including utilities and manufacturers. In addition, we considered the following stakeholders to define roles and responsibilities when implementing, testing and verifying the security measures:

- Administrators
- Application Developers
- Customers
- Device Manufacturers
- Utilities

- Regulation Authorities
- API Designers
- Security Researchers
- Standardization Bodies
- Energy Suppliers
- Software Developers and Providers
- Independent Test Institutions
- Maintenance Personnel

To increase practical applicability, the following *best practice* risk mitigation strategies have been defined under consideration of the stakeholder roles and common risks in today's and near future smart grid infrastructures:

Authenticity and Integrity

Device manufacturers and *software providers* need to cryptographically sign their software so that *customers* can verify the authenticity and integrity of the software product. *Customers* need an established verification process for cryptographic software verification. *Application developers* have to ensure that any communication within the application and the smart grid infrastructure is encrypted, authenticated and integrity protected. *API designers* need to employ cryptographic authentication protocols such as certificate based authentication schemes to ensure at any time that the communication occurs only between authorized senders and receivers. *Administrators* and *maintenance personnel* need established methods to verify the authenticity and integrity of communication channels and employed software products.

Security Evaluation, Penetration Testing and Interoperability

Device manufacturers need to follow a secure development life-cycle from early design phase to product rollout and maintenance. The life-cycle needs to include external security audits prior to product rollout. *Security researchers* and *independent test institutions* need to carry out penetration tests. Identified vulnerabilities need to be communicated to the *device manufacturer* and disclosed to the public by following a responsible disclosure process. *Utilities* need to carry out interoperability tests in a realistic lab environment. Both, *utilities* and *energy suppliers* need to consider the results of security audits in their product choice and need to carry our independent security audits in their own environments as well.

Minimum Attack Surface

Device manufacturers need to minimize the attack surface on their interfaces. Communication should be only established if really necessary. If so, any communication needs to be encrypted, authenticated and integrity protected per default configuration (i.e., secure-by-default). Unnecessary functions and functions not suitable for production use (e.g., testing functions) need to be removed or disabled. *Utilities* need to configure their systems in a way that only necessary functions and interfaces are enabled and no unnecessary data is generated. Security critical data needs to be cryptographically protected according to the current state-of-the-art and should be accessible by authorized personnel only (need-to-know basis). *Energy suppliers* and *utilities* need to be careful in their product choice and configuration that disabled functions can not be re-enabled through minimal manipulations, new firmware can not be easily uploaded and data is only stored locally and cryptographically protected whenever possible. *Application developers* need to be careful to keep data local whenever possible. If data needs to be transferred over communication links, communication protocols and subsequent data storage have to rely on established and practically proven protections mechanisms with proper data encapsulation, encryption, authentication and integrity protection.

Network Segmentation and Compartmentalization

Device manufacturers ensure in their product design that their products are only connected to the Internet when absolutely necessary. Devices should be shipped with up-to-date security patches, regular users should not be able to use administrative accounts, devices should be located behind firewalls and security critical actions should be logged and comprehensible. In addition, the device implementation should follow a layered security model including techniques such as virtualization or jailing so that in case of a breach of one security layer, the other security layers still prevent the security breach of the overall system. *Utilities* and *energy suppliers* need to establish the same principles in their own systems. *Administrators* need to ensure that these security measures are rolled out, documented and continuously maintained. *Maintenance personnel* has to continuously verify that the security measures are in place and working as expected. Any irregularities or incidents need to be reported so that appropriate action can be taken.

Remote Access and Compulsory Manufacturer Transparency

Device manufacturers ensure that remote access and management functions rely on open, established and secure communication protocols. The communication channel and the data need to be cryptographically protected to guarantee confidentiality, authenticity and integrity. Any remote access activities need to be logged. *Utilities* and *energy suppliers* choose products with open, established and secure remote access and management functions. *Administrators* need to ensure that remote accesses are handed in a secure way and performed remote management actions are monitored. *Maintenance personnel* must conduct remote accesses from within a secure maintenance environment and have to report irregularities or incidents.

Security Requirements for Manufacturers

Security researchers and *independent test institutions* support in the definition of smart grid security requirements through the publication of research results and joint activities within standardization and regulatory bodies. *Standardization bodies* and *regulation authorities* define and enforce security requirements that need to be implemented by *device manufacturers*. *Utilities* and *energy suppliers* need to acquire products that fulfill these security requirements.

Change, Patch and Configuration Management

Device manufacturers and *software developers and providers* continuously release security patches for discovered vulnerabilities. *Utilities* and *energy suppliers* install the security patches and keep them up to date. Depending on the security patches, additional security audits by *independent test institutions* might be necessary to account for the incurring system changes. Patches and configuration changes are managed and tested in realistic lab environments before roll-out in the field.

Physical Security

Device manufacturers need to consider physical and implementation attacks on their devices. Devices should be implemented tamper-resistant so that unauthorized physical access to the device and its interfaces is impeded. If applicable, countermeasures such as the partial disabling of security critical functions should be implemented. Tamper attempts should be logged in a secure non-volatile way and alarms should be communicated to the system operator such as the *utility* or the *energy supplier*. *Utilities* and *energy suppliers* monitor their devices in the field accordingly. *Administrators* need to handle detected incidents.

Incident Management

Device manufacturers include mechanisms in their implementations to detect security incidents on the device. Incidents are communicated to the system operators and local countermeasures can be taken. *Utilities* and *energy suppliers* have intrusion detection systems (IDS) or intrusion prevention systems (IPS) in place. If security incidents are detected, there need to be established incident handling procedures in place so that security incidents can be handled and mitigated in a timely manner.

In the subsequent compliance checking step, it needs to be ensured that an overall high level of security is continuously maintained. Previously identified and already mitigated threats thus need to be tracked and system components must be checked automatically on a regular basis. This way, if a components is added to the system that contains a known vulnerability (i.e. due to an old firmware version), the automated approach ensures that the components are identified early and the already established measures such as firmware upgrades can be taken to mitigate known risks. Although risks needs to be covered on an individual basis, a description of our generic smart grid risk mitigation approach is provided in Section 3.2. The (individual) risk mitigation step thus closes the security management cycle visible in Figure 1.34.

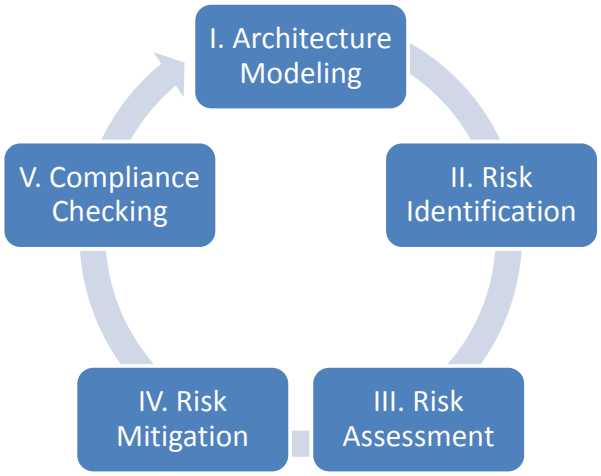

Figure 1.34: Architecture-Driven Smart Grid Risk Management Approach

1.7 Scientific Contribution

Architecture Driven Smart Grid Risk Management

We contribute a cumulative architecture driven smart grid risk management approach based on the Smart Grid Architecture Model (SGAM) [103] that represents both current and near-future European smart grids. On top of our smart grid modeling approach, we designed a threat catalog and an accompanying practical risk management approach that has been refined and evaluated through expert interviews with utility providers and manufacturers. Our smart grid risk management approach has been published and presented at the *3rd International Conference on Smart Grids and Green IT Systems (SMARTGREENS)* in 2014 [52] and at the *2nd ACM Workshop on Information Hiding and Multimedia Security* in 2014 [51].

Physical Attacks on Embedded Smart Grid Devices

We contribute an evaluation of physical attacks against embedded smart grid devices and present practically usable silicon reverse engineering techniques to extract the secret firmware from proprietary devices through silicon test modes. Our evaluation and the silicon reverse engineering approach have been published in the book *Smart Grid Security: Innovative Solutions for a Modernized Grid* in 2015 [99] and at the *21st ACM Conference on Computer and Communications Security (ACM CCS)* in 2014 [54].

Peripheral Proxying to Enable Dynamic Firmware Security Analysis

To address the challenge of dynamic firmware security analysis, we contribute PROSPECT, a transparent proxy for tunneling peripheral hardware accesses from the embedded system under test to a virtual analysis environment. In addition to PROSPECT, we evaluated the use of peripheral device communication caching with firmware program state approximation as potential solution to enable powerful firmware analysis techniques such as snapshotting, test parallelization or testing without physical access to the embedded system under test. Our contributions have been presented and published at the *9th ACM Symposium on Information, Computer and Communications Security (ASIACCS)* in 2014 [56] and the *10th International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)* in 2016 [50].

Efficient High Speed Attacks Using FPGA Clustering

To assess the practical security of cryptographic authentication mechanisms, we analyzed the well known WPA2-Personal authentication protocol and contribute a highly optimized design and architecture of a scalable FPGA-cluster based system for brute force attacks. On the same hardware, our implementation is 5 times as fast as the currently marketed world's fastest FPGA-based WPA2 password recovery system and has been published at the *Workshop on Cryptographic Hardware and Embedded Systems (CHES)* in 2016 [55].

1.8 Conclusion and Future Work

In the following sections, we present a holistic embedded system security analysis approach under special consideration of critical infrastructures such as the smart grid. In Chapter 2, we address the problem of smart grid risk assessment with an architecture driven approach that uses Smart Grid Architecture Model (SGAM) modeling [103] to identify high risk components in both current and near-future European smart grids. Utilities utilizing our architecture driven smart grid risk assessment approach can thus identify high risk components and select them for embedded security audits. In Chapter 3, we extend the idea of architecture driven smart grid security assessment towards a holistic smart grid security management approach allowing utilities to increase and maintain the security of their smart grid installations. In Chapter 4, we evaluate the use of physical attacks on smart grid components. However, while those physical attacks can be used by malicious attackers, some of these attack can also aid security analysts to extract firmware from smart grid components. Having access to the firmware is thus one of the basic requirements to conduct independent embedded security audits and to subsequently discover critical vulnerabilities deeply hidden in the firmware of today's embedded critical infrastructure devices. In Chapter 5, we explore the use of silicon reverse engineering techniques in case more traditional firmware extraction techniques such as the use of common debug and programming ports fail. We show that limited silicon reverse engineering is a viable way to discover test modes hidden in silicon that often allow the extraction of the firmware. Once the firmware is accessible to the security analyst, in Chapter 6, we present how peripheral device proxying can be utilized to overcome many of the current dynamic analysis challenged for embedded devices. Our system PROSPECT allows analysts to execute the device firmware in a virtualized analysis environment while the peripheral devices on the target device can still be accessed by the firmware under analysis. In Chapter 7, we extend the idea of PROSPECT and explore the use of a peripheral device communication cache with the help of program state approximation as a potential solution to enable more powerful firmware analysis techniques such as snapshotting, test parallelization or testing without physical access to the embedded system under test. In addition to firmware extraction and subsequently firmware security testing, security analysts need effective methods to test the practical security of cryptographic authentication mechanisms employed on embedded systems in the field. In Chapter 8, we thus present the design, architecture and implementation of a low-cost FPGA cluster based system that can perform high speed brute-force attacks on systems such as the well known WPA2-personal cryptographic authentication protocol that was the target of our implementation. In the future, we plan to extend our research in the field of implementation attacks with special focus on powerful invasive attacks such as IC deprocessing, computer aided IC reverse engineering, IC modification utilizing a Focused Ion Beam (FIB) and probing attacks.

CHAPTER 2

Practical Risk Assessment Using a Cumulative Smart Grid Model

Over the last years, the electrical power grid has undergone a tremendous change. The traditional power grid could be described as a producer-consumer model. The producers generate electricity and the electricity is transferred by utilities to the consumers. As a result, the amount of employed ICT technologies was limited. Today, there is a strong trend in the direction of sustainable green energy, energy saving and higher efficiency. Energy is no longer only produced at the top and delivered to the bottom; instead, everyone can become an energy producer. Consumers change to "prosumers" by running their own solar or wind power stations. Businesses and communities specializing on independent energy production through wind turbines, heating, or biogas plants emerge and grow. The boundaries in the traditional power grid model start to fade. On the other hand, large-scale energy producers and utilities can save energy and achieve higher energy efficiency by having the ability to influence or control devices in the user domain. To make this possible, energy grids are heavily expanded with ICT technologies – the traditional power grid is being transformed into the *smart grid*. On the downside, these technologies bear unforeseen risks for critical infrastructures. Smart grid ICT technology providers and utilities have limited experience with these new technologies and market pressure may force them to throw new products on the market before they have undergone quality assurance processes suitable for critical infrastructures. While traditionally access to smart grid ICT networks was limited to energy producers and utilities, new smart grid ICT technologies allow the massive amount of consumers to participate in these networks. Communication infrastructures in energy grids and especially in power grids are thus also exposed to a wide range of potential adversaries. To mitigate the security risks involved, there have been significant international efforts in terms of smart grid cyber security standards, risk assessment and security mechanisms (see Section 2.1). However, focusing on smart grids in the European Union and especially in Austria, it quickly turned out that many architectural or technological assumptions do not hold for the smart grid systems currently being rolled out in large quantities. For instance, within the European Union, the Smart Metering Protection Profiles [15, 16] are widely known for their security requirements

and definitions. Yet, smart metering is only one of many areas within the smart grid, and today's advanced metering infrastructure (AMI) typically does not correspond to the *gateway* and *security module* design concept suggested in those Protection Profiles.
As a result, in the Smart Grid Security Guidance $(SG)^2$ joint project [2] with leading smart grid component manufacturers and utilities, we set out to create a cumulative smart grid landscape model representing both current and future European smart grids. Based on established industry security standards and in close cooperation with manufacturers and utilities, we identified a comprehensive set of threats that are applicable within our cumulative smart grid model. To foster practical usability, we clustered both identified threats and smart grid systems to form the two dimensions of a threat matrix. This threat matrix allows practical threat assessment for both current and future European smart grids, and forms the basis for an according risk assessment determined by probability and impact. In summary, the main contributions of our work are:

- A cumulative smart grid model representing both current and near-future European smart grids as a basis for sound risk assessment
- A thoroughly designed threat catalog for modern smart grid architectures
- An accompanying practical risk assessment approach evaluated and refined in course of expert interviews with utility providers and manufacturers

The remainder of this paper is organized as follows. Section 2.1 provides an overview of related work. In Section 2.2 we explain how we developed the cumulative smart grid model. In Section 2.3 we describe our risk assessment approach, while Section 2.4 covers the evaluation and our results. The conclusions and suggestions on further work can be found in Section 2.5.

2.1 State-of-the-Art and Related Work

Mainly focusing on U.S. smart grids and technology, NIST has developed Guidelines for Smart Grid Cybersecurity [81]. In Europe, the German Federal Office for Information Security (BSI) has come up with a Common Criteria Protection Profile for the Gateway of a Smart Metering System and its Security Module [15, 16]. In contrast to our approach, the NIST guidelines do not allow an integrated approach. They are based on technologies employed in U.S. smart grids and give high-level recommendations only. Similarly, the BSI protection profiles do not provide a holistic approach either. Instead, they focus on smart metering only (which is only one building block of a smart grid), and their Target of Evaluation is a very specific smart metering implementation that does not reflect deployed smart metering systems.
Regarding risk assessment, Lu et al. outline security threats in the smart grid [72]. In comparison, our approach is targeted on the broad range of system components in European smart grids. Ray et al. provide a more formal approach to smart grid risk management [92] while one of our goals was to develop a practical risk assessment approach usable for utilities. Varaiya et al. show various ways to manage security critical energy systems [117]. However, they rather focus on formal methods, and their smart grid model does not reflect current European smart grids. Finally, Hou et al. outline the differences in risk modeling between traditional grids and smart grids [44],

while we focus on the smart grid landscape that is currently deployed or will be deployed in the near future. In addition, existing risk assessment approaches are covered in more detail in Section 2.3. Regarding smart grid security mechanisms, Yan et al., Mohan et al. and Vigo et al. provide an overview of security mechanisms for smart grids and smart meters [77, 120, 130]. While their work provides an overview of how security mechanisms should be realized, in our approach, we focus on the security mechanisms that are either implemented in current implementations or will be part of near-future implementations. Finally, Wang et al. [133] present smart grid standards covering security, but rather target U.S. standards like those published by NIST.

2.2 Cumulative Smart Grid Modeling using SGAM

In our joint approach to identify technological risks in current and future European smart grids, it turned out that leading experts, utilities and even manufacturers have a very different view and definition of what *the smart grid* is. Focusing on European smart grids and the Austrian power grid in particular, on a high-level view, the smart grid domains and actors within those domains are comparable to existing standards such as the NIST Smart Grid Framework [80] or the European Smart Grid Reference Architecture [103]. In a first task, we asked utilities to compare their deployed smart grid systems, model regions, pilot projects and concepts with existing reference models. Since, unlike the NIST framework, the European smart grid reference architecture focuses on European smart grid technologies, it was chosen as a basis for the comparison. Specifically, we used the Smart Grid Architecture Model (SGAM) [103] to allow for a well-structured comparison. Overall, 45 different projects could be identified and prioritized according to project size, project relevance, and both amount and quality of available information. The study showed that, in general, the SGAM model is usable with some limitations, but the reference model is only applicable on a high level. While it sketches the general structure of European smart grids, it does not contain detailed information on the technologies implementing smart grid components in this structure. For that matter, it is not adequate for qualified risk modeling suitable for utilities. To close this gap, we combined seven national and four international projects within the SGAM model to form a cumulative architecture model allowing us to deduce threats and risks for both current and future smart grid installations in Europe. The following sections describe our approach in more detail.

The Smart Grid Architecture Model (SGAM) Framework

The SGAM model had its original motivation in identifying gaps in standardization and locating these gaps in the SGAM model space. The model is structured in zones and domains (see Fig. 2.1). While the *zones* are derived from the typical layers of a hierarchical automation system (from field via process, station towards operation and enterprise level [93]), the *domains* reflect power-system specific fields of different actors such as transmission system operators, distribution system operators, and customers. In contrast to the NIST model, the European approach has a dedicated DER (Distributed Energy Resources) domain, which captures small distributed generators with their special infrastructure. Finally, in the third dimension, SGAM features *interoperability layers*. With these layers, the different aspects of networked smart grid systems are aligned. The base

layer is the component layer, where physical and software components are situated. On top of that, communication links and protocols between these components can be placed. The information layer holds the data models of the information exchanged. On top of that, the function layer holds the actual functionalities, and the uppermost layer describes the business goals of the system.
Today, SGAM serves three major purposes: first of all, it is a means to visualize and compare different smart grid automation architectures. This also allows the identification of gaps in all layers. Finally, SGAM can serve as a useful model to support model-driven architecture development. In this work, SGAM was applied for the first two purposes: comparison and identification of gaps.

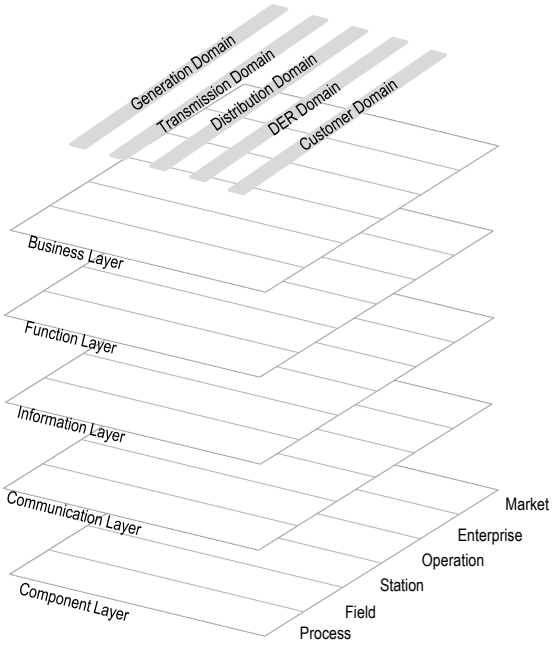

Figure 2.1: Smart Grid Architecture Model (SGAM) Framework

Current and Future Smart Grid Technologies within SGAM

Within the project, a holistic architecture was derived, which reflects the short- to mid-term extension of today's power grid IT technology towards future smart grid functionalities. The methodology used to achieve this architecture is based on individual SGAM modeling of national and international smart grid projects that significantly build on the use of ICT systems. From 45 project candidates, seven significant national smart grid research projects were selected for modeling:

- IEM: Intelligent Energy Management
- Smart Web Grids [105]
- DG DemoNetz Smart LV Grids [62]

- ZUQDE: Zentrale Spannungs- und Blindleistungsregelung mit dezentralen Einspeisungen in der Demoregion Salzburg [104]
- EMPORA: E-Mobile Power Austria [29]
- AMIS Smart Metering Rollout

A similar approach was applied to international projects. The selected significant projects were:

- The European FP7 Project OpenNode [87]
- The European FP7 Project EcoGrid EU [30]
- The US Demand Response Automation Server (DRAS) [26]
- The German ICT Gateway Approach OGEMA [86]

For SGAM modeling, detailed information about the technical implementations had to be requested from the projects under analysis. The availability of information (or contacts to the projects) was an additional selection criterion for the international projects.

European Smart Grid View according to the Cumulative SG Model

The derived architecture (Fig. 2.2) includes a harmonized cumulative view of the components found in all analyzed projects. The SGAM domains transmission and bulk generation were not covered by the selected projects since in the Austrian or, respectively, European view, the smart grid is primarily related to distribution systems. The architecture shows the component and communication layer of the SGAM model. On the bottom, the field devices can be found (such as smart meters or dedicated sensors and actuators). On the station level, both primary and secondary substation are situated with their current and prospective automation components. On the customer side, mainly residential customers, commercial buildings, and electric mobility can be found. On the top of the architecture, the enterprise level and market components are located. However, one of the main differences to existing models is the addition of detailed communication technology descriptions allowing a more in-depth risk assessment approach. The communication protocols and technologies underlined are the ones that are predominantly used in the projects we analyzed. For instance, in future smart grids the broad use of Web Services is anticipated for communicating with the energy market. Nevertheless, as depicted in our architecture, the predominant way to achieve this in current smart grids is to use personal communication via email or phone calls. From a risk assessment perspective, this results in a significant difference as Web Services can potentially be more easily compromised than a phone call made between a group of persons who probably know each other well from their daily work routine.

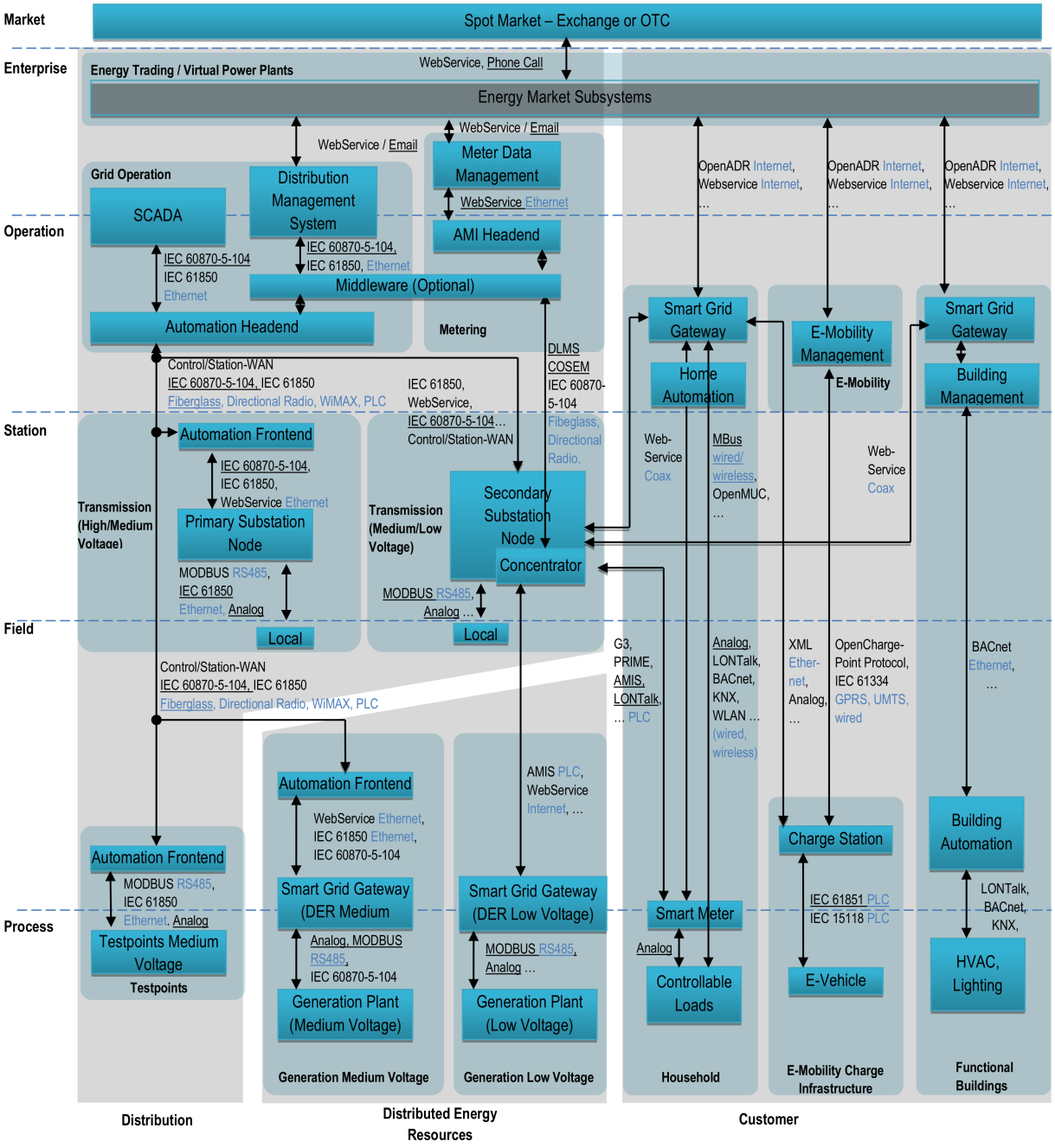

Figure 2.2: Cumulative Smart Grid Model

Evaluation of the Model

After a first draft of the architecture model had been developed, it was subject to a number of feedback rounds with members of the consortium (utilities and manufacturers). Improvements

and additional information were integrated into the architecture. The $(SG)^2$ architecture model serves as an anchor point for further analysis and as a common document of energy, IT and security experts. The main benefit of the model is that questions between the different expert domains can clearly be formulated and therefore easily be answered by referring to individual elements of the architecture.

2.3 Smart Grid Risk Assessment

Existing Approaches

Smart grid cyber security and risk assessment in particular has been addressed in several standards, guidelines and recommendations. The U.S. National Institute of Standards and Technology (NIST) has developed a three-volume report on "Guidelines for Smart Grid Cyber Security (NIST-IR 7628)" [81]: Volume two focuses on risks related to customer privacy in the smart grid, and gives high-level recommendations on how to mitigate these risks. However, no general approach for assessing security risks in the smart grid is provided. The European Network and Information Security Agency (ENISA) has issued a report on smart grid security. It builds on existing work like NIST-IR 7628 or ISO 27002 and provides a set of specific security measures for smart grid service providers, aimed at establishing a minimum level of cyber security [33]. Each security measure can be implemented at three different "sophistication levels", ranging from early-stage to advanced. The importance of a risk assessment to be performed before deciding the required sophistication levels is pointed out, but no specific risk assessment methodology is identified within the report.

The German Federal Office for Information Security has come up with a Common Criteria Protection Profile for the Gateway of a Smart Metering System and its Security Module [15, 16]. Both define minimum security requirements for the corresponding smart grid components based on a threat analysis. However, the Common Criteria approach, which focuses on a specific, well-defined Target of Evaluation, cannot provide a holistic view on cyber security threats in future smart grids. Another drawback is that the implementation of many smart metering systems currently being rolled out does not correspond to the defined gateway and security module design concept.

The CEN-CENELEC-ETSI Smart Grid Coordination Group has provided a comprehensive framework on smart grids in response to the EU Smart Grid Mandate M/490 [102]. As part of that framework, the "Smart Grid Information Security (SGIS)" report defines five SGIS Security Levels to assess the criticality of smart grid components by focusing on power loss caused by ICT systems failures.

Moreover, five SGIS Risk Impact Levels are defined that can be used to classify inherent risks in order to assess the importance of every asset of the smart grid provider. This means that the assessment is carried out under the assumption that no security controls whatsoever are in place. While this is a valuable approach, it is not suitable for a more practical scenario that focuses on actual, currently deployed or foreseeable implementations.

Risk assessment methodologies have also been addressed by the FP7 project EURACOM, which considered protection and resilience of energy supply in Europe and aimed at identifying a

common and holistic approach for risk assessment in the energy sector. As part of a project deliverable[1], existing risk assessment methodologies and good practices have been analyzed in order to identify a generic risk assessment method which could be customized to suit the specific needs of the energy sector. While most of the existing risk assessment methods are asset-driven, the $(SG)^2$ project required an architecture-driven approach for developing a risk catalog. This approach is described more closely in the following.

Our Approach: Threat Matrix and Risk Catalog

The risk assessment approach taken in $(SG)^2$ focused on the ICT architecture model initially developed within the project (see Section 2.2). The goal was to come up with a comprehensive catalog of ICT-related risks for smart grids in Europe from a Distribution System Operator's perspective. The following steps were taken to achieve that goal:

1. Compile a *threat catalog* for smart grids focusing on ICT-related threats and vulnerabilities
2. Develop a *threat matrix* by applying the threat catalog to the ICT architecture model, i.e., identify which threats apply to which components of the model
3. Assess the potential risk for each element within the threat matrix by estimating the probability and the impact of an according attack, thus eventually producing a *risk catalog*

These steps are explained in more detail in the following paragraphs.

Compiling the Threat Catalog

The $(SG)^2$ threat catalog was not to be developed from scratch, but should build upon a well-established source of ICT-related security threats. To that end, the IT Baseline Protection Catalogs developed by the Federal Office for Information Security [14] were chosen to form the main source of input as they provide a comprehensive list of security threats that could possibly apply to an ICT-supported system. Additionally, the threats specified in the smart-grid-specific Protection Profiles [15, 16] were taken into account. Non-technical threats, i.e., threats related to organizational issues or force majeure, were not considered due to the scope and focus of the $(SG)^2$ project. Thus, out of a list of initially 500 threats accumulated from the identified sources, the ones without any relevance to smart grids or without any relation to ICT were eliminated at first, yielding roughly half of the initial threats for further consideration. While certain threats listed in the BSI Catalogs are very generic, others apply to very specific settings only; therefore, it was necessary to merge some of the threats that remained in our list after the "weeding" step.

[1] The EURACOM project deliverables can be downloaded at `http://www.eos-eu.com/?Page=euracom`.

This resulted in a list of 31 threats, which were grouped into the following clusters:

- Authentification / Authorization
- Cryptography / Confidentiality
- Integrity / Availability
- Missing / Inadequate Security Controls
- Internal / External Interfaces
- Maintenance / System Status

Since the BSI Baseline Protection Catalogs are not tailored for any specific use case, the relevant threats had to be adapted to the smart grid scenario, i.e., they were interpreted in the smart grid context.

Developing the Threat Matrix

The next step on the path to the $(SG)^2$ risk catalog was to apply the threats identified in the first step to the components of the $(SG)^2$ architecture model (see Section 2.2), i.e., to state which threats are relevant for which of the components and why. To answer that question, the functionality and the characteristics of the individual architecture components had to be assessed first. For feasibility reasons, the granularity of the components to be considered was set to the level of the boxes depicted in dark grey in Fig. 2.2:

- Functional Buildings
- E-Mobility & Charge Infrastructure
- Household
- Generation Low Voltage
- Generation Medium Voltage
- Testpoints
- Transmission (High/Medium Voltage)
- Transmission (Medium/Low Voltage)
- Grid Operation
- Metering

The Energy Markets domain was not considered due to lack of current ICT utilization and lack of information on future functionalities.
For each element of the threat matrix, it was first decided whether the threat could be relevant to that particular component or not. If potential relevance was assessed, the reason for that decision was noted, and possible attack scenarios were developed. Since the architecture model considered also smart grid developments for the near future, reasonable assumptions regarding the implementation had to be made in certain cases. These assumptions were discussed and verified by the involved manufacturers and utilities.

Assessing the Risk Potential

The final step of the $(SG)^2$ risk assessment involved estimating the risk potential for each element of the threat matrix, thus providing a comprehensive risk catalog. To that end, the probability and the impact of each of the threats occuring was rated for each of the components of the architecture model. A semi-quantitative approach was chosen for the risk assessment, which had previously been applied and practically assessed by one of the member organizations of the project consortium: both probability and impact were measured on a five-level scale ranging from very low (level 1) to very high (level 5). The probability level was determined by the number of successful attacks per year, ranging from less than 0.1 incidents (level 1) to multiple incidents (level 5) per year. The impact of a successful attack was determined by monetary loss, customer impact, and geographic range of the effects (e.g., local, regional, global). The outcome of this step is a comprehensive catalog of cyber security risks on smart grids in Europe. The steps taken to evaluate the catalog as well as the main findings are described in the following section.

2.4 Evaluation and Results

A good threat and risk assessment model delivers useful results, i.e., captures specific threats appropriately, is easy to use, and well applicable in reality. With these targets in mind, we evaluated and revised the model in a series of workshops, where domain experts from both areas of information technology and energy rated the proposed risk assessment approach.

Evaluation Methodology

The evaluation, partly integrated in the overall creation of our cumulative smart grid model for risk assessment, included in the following three steps:

- *Step 1: Evaluation of Threat Catalog.* In order to evaluate our threat catalog, we set up end user workshops with experts from utility providers, device manufacturers, and academic institutions to evaluate both the relevance and completeness of the identified threats. During that evaluation, additional threats were added that are unique to the smart grid domain.
- *Step 2: Threat Relevance.* Experts surveyed the applicability of identified threats to the various domains in the SGAM model. This step enabled the identification of the most important threats per domain as well as their interdependencies, and further allowed to

focus on most relevant threats. The result was again discussed and refined with major utility providers in Austria in end user centric workshops.

- *Step 3: Probability and Impact Assessment.* In a last step, we had experts independently rate the probability of occurrence of identified threats and the impact from their point of view. These experts represent the opinions of different utility providers, ranging from small and locally operating organizations, to larger ones. In a joint workshop, the individual results were again discussed and consolidated to ensure the broad use of the resulting threat and risk catalog.

Main Findings

Dealing with proper risk assessment in the smart grid domain is indeed challenging, mainly because of the novelty, complexity, and multidisciplinarity of this topic. Here, we present some of the main findings, derived from the application of our approach in a real user context. We foresee these qualitative statements as a major contribution for future improvements of our smart grid risk assessment approach.

Unbalanced Risk Distribution

Essentially, the probability of a security breach is comparatively low on the upper levels of the cumulative smart grid model, because components are small in numbers and easy to protect. An attacker typically has no physical access to components, and well-trained experts acting under rigid policies maintain the grid's backend. Furthermore, protection mechanisms on the upper levels do not suffer from cost pressure on this level; for instance, redundant systems and hot stand-by sites are state-of-the-art here. However, once an attacker manages to get access to critical systems, the negative impact will eventually be high; for instance, shutting down a primary substation node could affect whole city districts. This makes the security of higher level smart grid components a first priority for utility providers. On the other hand, the probability of a security incident on the bottom level of the cumulative smart grid model is much higher. The reason is that attackers can easily get hold of components (e.g., a concentrator or a secondary substation node) or have them installed on their own premises (e.g., a smart meter). The impact of an attack towards these components is expected to be (geographically) limited at a first glance. The situation may however change tremendously if someone publishes a successful smart meter mass attack on the Internet. This could lead to unanticipated cascading effects in the power grid. Hence, smart grid security on the lower levels of the smart grid is of major importance for utility providers as well.

Evolution of the Grid

Security challenges arise from the fact that the current power grid architecture is just step-wise transformed to a smart grid. While neither a pure legacy system nor a completely new system designed from scratch is too hard to be properly secured, while a mixture of both *is*. We identified that during the transmission of the current power grid into a smart grid, we are facing many security challenges: (i) the mix of legacy protocols and new protocols; (ii) the usage of wrappers,

data converters, and gateways to make devices interoperable; (iii) short innovation cycles and rapid pace with which technologies advance clash with the traditional views of grid investments, where components have been designed to last for decades.

Technological Diversity

Not only the transformation phase, but also the final smart grid architecture foresees the application of a wide variety of different technologies [98]. Many security specialists argue that this diversity leads to a large attack surface, meaning that in hundreds of different protocols and implementations, a security relevant flaw is much more likely compared to only a small set of well-tested standardized technologies. This technological diversity and the need for seamless interoperability mostly avoids rigid designs and the setup of a uniform and secure architecture. On the other side, systems may be designed with different goals in mind so that the intermediary interfaces between them may lead to security vulnerabilities. Although standardization bodies, such as the NIST and BSI, have published (vendor-independent) viable technology recommendations and guidelines, there are no obligations for device vendors and utility providers to stick to them. However, we must not negate the positive aspects of technological diversity. Combining different technologies and products can help to prevent cascading effects caused by a specific vulnerability prevalent in a single technology or series of devices. As a consequence, an attack that has been successful at one utility provider is not necessarily successful at another one, if different technologies are deployed.

Risk Assessment Complexity

The complexity of risk assessment in the energy domain increases rapidly with the introduction of ICT components. This situation will become even worse, once smart grid technology is rolled out on a large scale, because systems become more and more coupled, even across geographical borders, resulting in strong interdependencies among components. The utility providers' concerns are therefore mainly centered around the understanding, detection, and mitigation of complex attack scenarios, where similarly to highly distributed computer systems, a multitude of vulnerabilities may be exploited in course of a complex attack. Estimating risks for such cases is extremely difficult, since numerous mostly unknown variables need to be considered. An example for such a complex attack case is the potential intrusion of malicious users into the metering backend through an exploitation of the smart meter communication network. Here, we argue that our architecture model, which clearly documents existing communication links as well as utilized protocols, is of significant help.

2.5 Conclusion and Future Work

In this work we analyzed existing smart grid standards with respect to smart grid security and risk assessment together with leading manufacturers and utilities. Our study indicated that standards like NIST-IR 7628 [81] or the Protection Profiles published by the The German Federal Office for Information Security [15, 16] are only of limited practical use to utilities. As a consequence, in a joint approach with leading manufacturers and utilities, we analyzed seven national and four

international smart grid projects to form a cumulative smart grid model representing both current and future European smart grids. In comparison to existing models like the U.S. NIST Smart Grid Framework [80] or the European Smart Grid Reference Architecture [103], our model is technically more detailed and includes a description of predominant communication technologies. In the second part of our work, we developed an extensive methodology to assess the risks involved within our cumulative smart grid model. While our threat catalog initially comprised a list of more than 500 threats, we were able to create a clustered catalog of no more than 31 threats that can be practically handled on top of the smart grid areas in our model. Due to the close cooperation with leading manufacturers and utilities, we believe that our work has a high practical impact on European utilities, as it can support them to conduct a risk analysis of their specific infrastructure.
In near future, we also plan to extend our risk catalog with respect to risk mitigation approaches and security controls, so that utilities can effectively mitigate identified risks with the help of our framework.

2.6 Acknowledgements

This work has been made possible through the joint SG^2 KIRAS project under national FFG grant number 836276.

Threat Category	Threat	Functional Buildings	E-Mobility incl. Infrastruct.	Household	Generation (Low Volt.)	Generation (High Volt.)
Authentication / Authorisation	*Defective or missing authentication or inappropriate handling of authentication data*	Relevant for all interfaces to the Smart Grid Gateway (Building Automation, Market/Internet und Secondary Substation) (P: 2; I: 2)	An attacker could impersonate the EMS and take control over all charge stations in an affected area (e.g., a street), which could lead to instabilities in the grid (P: 2; I: 3)	Relevant for Smart Grid Gateway and Smart Meter and all interfaces (configuration interface, Market, PLC to data concentrator and Secondary Substation) (P: 2; I: 2)	Exact scope and functionality of Smart Grid Gateway is not clear to date - how will the authentication towards the Automation Front-end and the Smart Grid Gateway be handled? (P: 1-3; I: 2-3)	Exact scope and functionality of Smart Grid Gateway is not clear to date - how will the authentication towards the Automation Front-end and the Smart Grid Gateway be handled? (P: 1-3; I: 3-4)
Cryptography / Confidentiality	*Disclosure of sensitive data*	Building automation data are sensitive since they may give away information about productivity or working hours (P: 2; I: 2)	Confidential load data could be eavesdropped on through the connection to the E-Mobility Management System or Smart Grid Gateway (P: 2-3; I: 2)	Metering data is confidential since it affects privacy and habits of the customers; impact of accidental disclosure depends on the scope of data and the number of households affected (P: 2; I: 3)	Output data of small producers is confidential since they may allow conclusions to be drawn on consumer behavior; high probability since no protection measures currently in place (P: 4; I: 3)	Hardly relevant since outout data are not confidential; low motivation (P: 1; I: 1)
Integrity / Availability	*Tampering with devices*	Hardly relevant due to physical access control; Smart Grid Gateway could be physically part of the Building Automation system; building operator has no motivation to compromise his own load management (P: 1; I: 1)	Tampering with EV, charge station or E-Mobility Management System; a malicious user can easily access relevant components; impact depends on whether private or public charge station is targeted (P: 3; I: 2-3)	Customers have direct access to the components; main motivation is fraud, but more severe attacks might as well be possible (e.g. issuing control commands) (P: 4; I: 3)	Relatively exposed (in a household); the amount of energy supplied may be a subject to fraud; depends on the use of anomaly detection techniques and plausibility checks (P: 4; I: 2)	Basic level of physical protection; low probability due to professional operator (P: 1; I: 4)
Missing / Inadequate Security Controls	*Defective or missing security controls in networks*	Communication via the network in the building; Smart Grid Gateway communicates via insecure networks (Internet), which opens up the possibility of distributed attacks on the Smart Grid Gateway (P: 1; I: 4)	Use of protocols that lack security features (e.g. IEC 61334 via PLC)could lead to eavesdropping on confidential load data; negative consequences for manufacturer / operator of the charge station, although the utility may also be affected (P: 2; I: 2)	Communication over insecure networks (Internet, PLC) (P: 3; I: 3)	Especially relevant for Internet connection to Secondary Substation Node (P: 2; I: 2)	Especially relevant for communication over the telecontrol WAN, currently IEC 60870-5-104 without encryption (P: 2; I: 3)
Internal / External Interfaces	*Illegal logical interfaces*	Illegal communication channels between the Gateway and interfaces (Market/Internet, Secondary Substation, Building Automation) could be established and exploited in an attack (P: 2; I: 3)	Illegal communicating with EV, charge station or E-Mobility Management System could lead to disclosure of confidential load data (P: 2; I: 3)	Illegal communication channels between the Smart Grid Gateway and ist interfaces (Market/Internet, Secondary Substation, Smart Meter) could be establish and exploited in an attack (P: 2; I: 3)	Due to physical exponation an attacker could interact with the system directly, thus revealing new interfaces that could be exploited (P: 3; I: 2)	Basic level of physical protection; access to telecontrol WAN may be feasible through faulty operation or targeted attacks (P: 2; I: 3)
Maintenance / System Status	*Operation of unregistered or insecure components or components with overly broad range of functions*	The Gateway could have a wide range of capabilities and functions, thus increasing the attack surface via the interfaces (Building Automation, Market/Internet, Secondary Substation) (P: 2; I: 2)	Possible disruption of the E-Mobility Management System or Smart Grid Gateway if connected to a non-standard or manipulated charge station ; local impact only (P: 1; I: 1)	Smart Grid Gateway or Smart Meter could have a too wide range of capabilities and functions, thus increasing the attack surface via the interfaces (Smart Metering, Market, Secondary Substation) (P: 2; I: 2)	Rather low probability since the components are dedicated to a very specific use; however, backdoors constitute a realistic threat scenario especially in replicated devices (P: 3; I: 2)	Rather low probability since the components are dedicated to a very specific use; however, backdoors constitute a realistic threat scenario (P: 2; I: 3)

Table 2.1: Threat Assessment (Part 1). Notice, the threat category and an exemplary threat are given in the first two columns. Subsequent columns contain the quantitative and qualitative assessments results for (P)robability and (I)mpact on a scale from 1 to 5 for each threat and per domain.

Threat Category	Threat	Testpoints	Transmission (High/Med. Voltage)	Transmission (Med./Low Voltage)	Grid Operation	Metering
Authentication / authorisation	*Defective or missing authentication or inappropriate handling of authentication data*	No authentication between Automation Front-end and Testpoint, although the Front-end authenticates towards the Primary Subst. Node (PSN); a spoofing attack could lead to false data being sent to the PSN; low impact (suboptimal supply) (P: 2-3, I: 1)	Due to the relatively low number of Primary Substation Nodes accounts or parameters can be configured and tested manually, therefore the probability is lower than in lower parts of the architecture model (P: 1-2, I: 4)	Especially relevant for remote maintenance access points; Secondary Substation Node and Concentrator can be easily accessed mostly, which gives a high probability; possibly regional impacts in case of unauthorized access (P: 4; I: 3)	Access rights management on SCADA systems implemented on an individual basis at utilities, standard IT technologies are employed; ïdentity spoofingöf calls to EMS; main threat is exploitation of insecure remote access solutions (P: 2, I: 4)	The connection between AMI headend and Smart Meters is encrypted by utilizing strong cryptographic primitives (ECDSA, AES); authentication and authorization is realized through certificates; thus, a high security standard is expected (P: 1, I: 2)
Cryptography / Confidentiality	*Disclosure of sensitive data*	Does not apply since test data (voltage, frequency) is not confidential	Current supply data and control commands are probably not confidential; consumption data are aggregated, therefore low impact (P: 2, I: 1)	Processing of confidential supply/consumpt. data in Concentrator and Secondary Substation Node; medium probability due to little (physical) protection (P: 3, I: 3)	For load estimation, consumption and energy production data is anonymized; power grid plans and control data is required to be protected accordingly (P: 1-2, I: 4)	Relevant since confidential power consumption data is processed and stored (P: 1, I: 3)
Integrity / Availability	*Tampering with devices*	Hardly relevant due to physical access control mechanisms in place; manipulation via the communication interface could be feasible (e.g. changing the configuration on SD card); low impact (P: 2, I: 1)	Tampering hardly relevant due to high voltage, but this may not hold for malicious insiders; currently strong impact (outage), but timely mitigation can be expected (P: 2, I:3)	Tampering possible especially with Concentrator; rather high probability due to little (physical) protection measures; regional impact (P: 3-4, I: 3)	Relevant if direct manipulation by insiders; remote manipulation over telecontrol WAN possible; forged process data may cause malfunction of DMS and subsequently lead to grid instability (P: 2, I: 4)	Low relevance due to physical protection measures (P: 1, I: 3)
Missing / Inadequate Security Controls	*Defective or missing security controls in networks*	Connected to Primary Substation Node via telecontrol WAN with IEC 60870-5-104 (unencrypted); low impact (suboptimal supply) (P: 2-3, I: 1)	Especially relevant for the communication via the telecontrol WAN; probability is low since security controls in place are more efficient than for the components in the lower parts of the architecture model (P: 1, I: 1-2)	Especially relevant for Internet/PLC connection to Middleware and Smart Grid Gateway (P: 2-3, I: 3)	Relevant for some interfaces (telecontrol WAN, EMS link); link to PSN secured with IPSec; telecontrol WAN link to Secondary Subst. Node could include forged data causing grid instability; (P: 2, I: 4)	Especially relevant for the interfaces to the outside world (i.e. connection to concentrators) (P: 1, I: 2)
Internal / External Interfaces	*Illegal logical interfaces*	Relevant for unauthorized access via telecontrol WAN; a DoS-attack on the Primary Substation Node could lead to generation plants going offline, resulting in voltage drops (P: 2, I: 2-3)	Relevant for access via telecontrol WAN (P: 2, I: 4)	Relevant for access via telecontrol WAN (P: 2, I: 4)	Relevant due to unauthorized access over external interfaces: telecontrol-WAN (Automation Headend), Webservice (EMS), Remote Access (SCADA) depending on deployed technologies (P: 2, I: 4)	Due to illicit logical interface (e.g. due to a successful attack) a connection from the metering system over the middleware to the grid operation system is feasible (P: 1, I: 3)
Maintenance / System Status	*Operation of unregistered or insecure components or components with overly broad range of functions*	Unregistered components hardly relevant (physical access control); an overly broad range of functions possible despite on-site maintenance (mostly no remote maintenance); low impact (P: 1-2, I: 1)	Unregistered components hardly relevant due to physical access control; an overly broad range of functions possible; low probability due to highly specialized components (compared to home area) (P: 1, I: 2-3)	Due to easy accessibility of the substations unregistered components could be installed; regional impacts (P: 3-4, I: 3-4)	Unregistered components are of low relevance due to physical access protection; unused but active system functionalities are relevant, especially considering remote access or support interfaces for manufacturers (P: 2-3, I: 4)	Unused but active system functionalities in the AMI headend lead to an increased attack surface (P: 2, I: 2)

Table 2.2: Threat Assessment (Part 2). Notice, the threat category and an exemplary threat are given in the first two columns. Subsequent columns contain the quantitative and qualitative assessments results for (P)robability and (I)mpact on a scale from 1 to 5 for each threat and per domain.

CHAPTER 3

Architecture-Driven Smart Grid Security Management

While traditionally electrical power grids adhered to the producer-consumer model, in modern smart grids everyone can become an energy producer – by leveraging green energy produced through solar panels, wind turbines or heating and biogas plants, consumers turn into "prosumers". For traditional large-scale utilities and energy producers, this has introduced a massive drawback: due to decentralized energy production, energy networks can no longer be centrally controlled. The solution is to upgrade existing power grids to smart grids by establishing an ICT network in parallel to the electrical power grid. While this brings advantages with respect to energy efficiency, green energy harvesting and consumer freedom, it also introduces ICT security risks in critical infrastructures that may cause disastrous effects.

As manufacturers of smart grid components move from pure electrical systems to the development of complex ICT systems, and though security may be an important target for them, market pressure and a lack of security experience may force them to roll out insecure products. Utilities, on the other hand, need to rely on manufacturers that their smart grid devices are secure in order to run this critical infrastructure. To lower the risks involved, proper risk management needs to be put in place. However, existing ICT-related risk management processes are not directly applicable to the smart grid domain as the technology and the security requirements are significantly different. On the other hand, readily available smart grid security guidelines such as the Protection Profiles [15, 16] developed by the German Federal Office for Information Security (BSI) merely focus on smart metering and thus do not map to the entire smart grid architectures deployed.

We decided to take a different approach. Instead of focusing on a single technological component, we model European smart grid architectures by using the Smart Grid Architecture Model (SGAM) [103]. Based on well-established sources of ICT-related security threats, we created a catalog for ICT security threats to the smart grid, which can be applied to components in the SGAM model. In this work, we show how our approach can be practically used for smart grid risk management, including risk assessment, mitigation and compliance checking. As our approach

has been developed in conjunction with leading smart grid manufacturers and utilities, we believe that it has a strong practical impact.
In summary, the main contributions of our work are:

- an SGAM-based smart grid model representing both current and near-future European smart grid architectures,
- a comprehensive catalog of cyber security threats for smart grids, and
- a practical risk assessment approach able to bridge the gap between a high-level architectural view and specific technical security measures.

The remainder of this paper is organized as follows. Section 3.1 outlines existing work on smart grid security and risk assessment. Section 3.2 describes the five steps of our smart grid risk management approach, which is subsequently evaluated in Section 3.3. Section 3.4 concludes the paper and identifies potential areas for future work.

3.1 State-of-the-Art and Related Work

Smart grid technologies have received major attention in both academia and industry in recent years. Various works discuss the basics of the smart grid, such as its structure, application, and potential impact [3,111]. Others cover established and recently developed technical standards [25]. The European Union plans to replace traditional electricity meters with smart meters to a large extent until 2020, which draws major attention to various security and privacy aspects of this technology [60, 98]. Therefore, the U.S. NIST and European ENISA have released numerous guidelines on how to secure smart grid architectures [33, 81]. Although these documents build a solid basis, they do not show the complete picture. NIST, for instance, focuses on technologies employed in U.S. smart grids, and both guidelines give quite high-level recommendations only. Similarly, the BSI Protection Profiles [15, 16] do not provide a holistic approach either. Instead, they focus on smart metering only (which is only one building block of a smart grid), and their target of evaluation is a very specific smart metering implementation that does not reflect deployed smart metering systems.
The electric grid is perhaps the most critical infrastructure today, and thus safety, i.e., reliability and availability, is a top priority. Potential vulnerabilities of smart metering systems – and the grid in general – are widely discussed topics [60, 75, 122]. As a consequence, many research works focus on quite small (technical) parts of the overall smart grid architecture. For instance, data communication security controls (e.g., cryptographic functions such as encryption, message authentication codes, and digital signatures) provide standard security services in terms of confidentiality, integrity, and accountability of messages and their origin [25]. Others deal with effective key distribution [121] and management for devices with very limited computational power [75] to enable efficient encryption of meter readings and access control (similar to Pay-TV access control systems [121]). Yan et al., Mohan et al. and Vigo et al. provide an overview of security mechanisms for smart grids and smart meters [77, 120, 130]. While their work provides an overview of how security mechanisms should be realized, in our approach, we focus on

the security mechanisms that are either implemented currently or will be part of near-future implementations.
The Smart Grid Coordination Group formed by the European standards organizations CEN, CENELEC and ETSI has provided a comprehensive framework on smart grids in response to the EU Smart Grid Mandate M/490 [102]. As part of that framework, the "Smart Grid Information Security (SGIS)" report defines five SGIS *Security Levels* to assess the criticality of smart grid components. Additionally, five SGIS *Risk Impact Levels* are defined that can be used to classify inherent risks in order to assess the importance of every asset of the smart grid provider. The assessment is carried out under the assumption that no security controls whatsoever are in place. Compared to the work carried out by the SGIS group within M/490, our main goal was to develop a practical risk assessment approach for smart grid systems that are currently deployed or will be deployed in the near future. Our approach should be readily applicable by utilities in contrast to more formal approaches as suggested for example in [92, 117].

3.2 Smart Grid Risk Management Approach

Most efforts on smart grid security either deal with threats and vulnerabilities on an abstract, high architectural level, or focus on very specific technical aspects, e.g., encryption or authentication, without considering the overall picture.

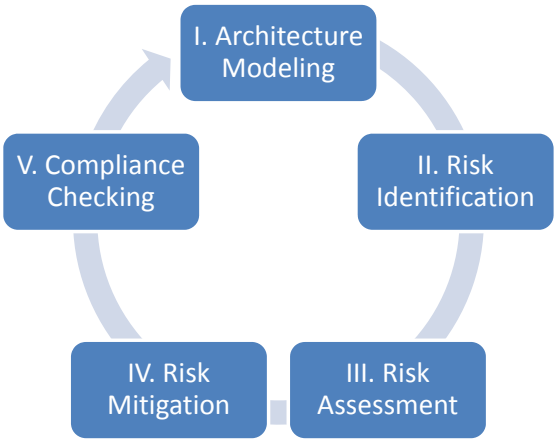

Figure 3.1: Architecture-driven Smart Grid Risk Management Approach

Our proposed smart grid risk management approach therefore aims at bridging the gap between a high-level architectural view and specific technical security measures. For that purpose, it employs a five-step cyclic process model depicted in Fig. 3.1, which consists of the following phases:

I. Architecture Modeling

II. Risk Identification

III. Risk Assessment

IV. Risk Mitigation

V. Compliance Checking

First, it allows DSOs (distribution systems operators) to map their deployed components to the standard architecture model SGAM. This phase is crucial to get a holistic view on the deployed components and their underlying technologies in a standardized and structured manner. The second phase subsequently enables a sophisticated risk identification and a later risk assessment. Based on the concrete technologies employed, specific technical controls (in addition to organizational measures) can be applied to mitigate the identified risk. If, for instance, the architectural model reveals insufficiently secured communication lines, potential technical mitigation measures are to use stronger authentication and encryption methods. Eventually, in the fifth phase, compliance to technological guidelines, regulations and corporate strategy needs to be ensured in order to avoid undesired secondary effects of mitigation measures. The whole model, from phase I to V, is cyclic since every mitigation action will eventually cause adaptations of the architecture, which need to be reflected in the model maintained in phase I. Following these phases, our approach is able to provide concrete technical solutions without losing a connection to the overall picture. In the following paragraphs, we explain each phase more closely.

Architecture Modeling using SGAM

In order to model smart grid architectures, we employ the Smart Grid Architecture Model (SGAM) [103]. The SGAM model was originally intended to identify standardization gaps in smart grid standardization processes. The model is structured in zones and domains. The *zones* are derived from hierarchical automation system models that classify systems into Field, Process and Station towards Operation, Enterprise and Market level [93]. The *domains* reflect power-grid-specific domains ranging from the Customer, Distributed Energy Resources (DER), Distribution and Transmission to the Generation domain. In contrast to the NIST Smart Grid Framework [80], SGAM features a dedicated DER domain, in which small distributed generators with their special infrastructure find their place. Finally, in the third dimension, SGAM has *interoperability layers* that highlight different aspects of networked smart grid systems from hard- and software components over communication links and protocols up to functional and business layers. We used SGAM as a means for visualizing and comparing different smart grid automation architectures and depicting existing and near-future smart grid architectures (see Fig. 3.2). A more detailed description of our architecture model can be found in [53].

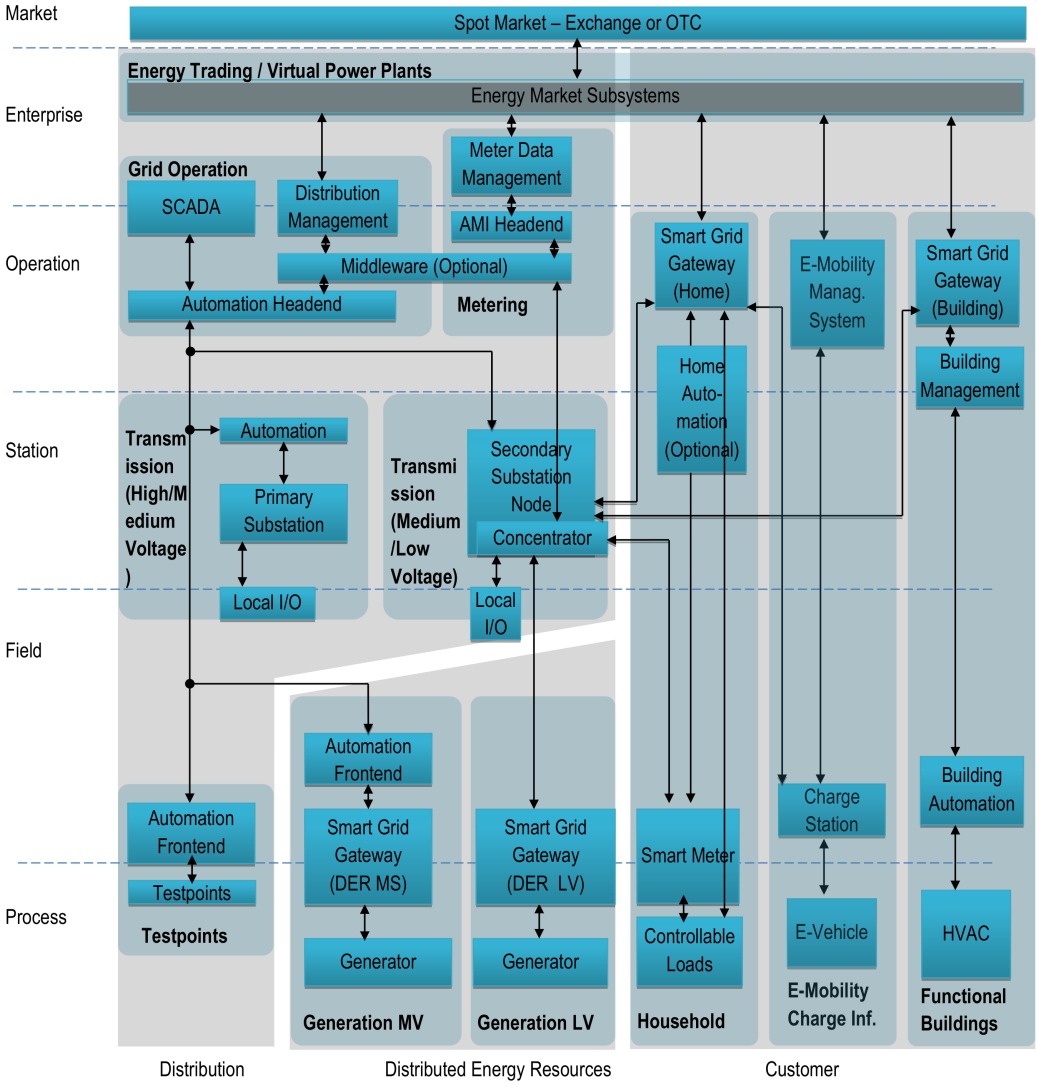

Figure 3.2: Simplified Proposed Architecture Model

Risk Identification

In order to identify risks that can occur within smart grid environments, we compiled a threat catalog focusing on technical threats. Since the threat catalog should build upon a well-established source of ICT-related security threats, we used the IT Baseline Protection Catalogs [14] as our main source. Threats quoted in the smart-grid-specific Protection Profiles [15, 16] were also taken into account. We focused on technical threats and thus omitted organizational threats or force majeure. All remaining threats were checked for their generic applicability in smart grid environments and filtered accordingly. As some of the threats in the BSI Catalogs are very specific while others are more generic, we adapted the threats to the smart grid scenario and merged them into a practically usable threat catalog comprising 31 threats (see Table 3.1).
These threats were subsequently interpreted in the smart grid context and grouped into the following clusters:

- Authentication / Authorization
- Confidentiality
- Integrity / Availability
- Internal / External Interfaces
- Maintenance / System Status
- Missing / Inadequate Security Controls

Since the threats in the threat catalog are kept in a generic form, there is no need to adapt the threat catalog in case the smart grid architecture model (see Section 3.2) changes.

Risk Assessment

In the next phase, the threats identified in phase II are applied to the architecture components which have been defined in phase I. For each component and threat, we evaluated both the likelihood as well as the impact of a threat to occur. Both probability and impact were measured on a five-level scale ranging from very low (level 1) to very high (level 5), depending on the frequency and range of successful attacks. However, while this could be exercised on all smart grid components in the SGAM model, it would quickly become impractical due to the high number of elements in the threat matrix. For this reason, we decided to cluster smart grid components into the following building blocks (cf. Figure 3.2):

- Functional Buildings
- E-Mobility & Charge Infrastructure
- Customer Premises
- Generation Low Voltage

Threat Category	Threat
Authentication / Authorization	Defective or missing authentication or inappropriate handling of authentication data Defective authorization
Confidentiality	Defective key management Disclosure of sensitive data Insecure encryption methods or parameters
Integrity / Availability	Outage or disruption of IT systems Outage or disruption of networks or network components Outage or disruption of supply networks Tampering with devices Tampering with data Loss or corruption of data due to physical factors Loss or corruption of data due to misuse or negligence Fee fraud
Internal / External Interfaces	Illegal physical interfaces Illegal logical interfaces Incompatibilities between systems or (network) components
Maintenance / System Status	Operation of unregistered or insecure components or components which provide unnecessary services Missing or inadequate maintenance Insufficient anomaly detection Insufficient dimensioning Security issues during software migration Insufficient monitoring and controlling capabilities Faulty use or administration of IT systems Faulty time synchronization Faulty data synchronization Uncontrolled cascading effects
Missing / Inadequate Security Controls	Defective or missing security controls in networks Defective or missing security controls in software products Software vulnerabilities or bugs Use of insecure protocols Failure or disruption of safety controls

Table 3.1: Threat Catalog

- Generation Medium Voltage
- Test Points
- Transmission (High/Medium Voltage)
- Transmission (Medium/Low Voltage)
- Grid Operation
- Metering

In case of considerably different smart grid architectures and models, these building blocks might differ and would need to be adapted accordingly. However, in the common case there is no need for adaptation due to the generic form of the threats and building blocks.

Probability \ Severity	1	1,5	2	2,5	3	3,5	4	4,5	5
1	1	1,5	2	2,5	3	3,5	4	4,5	5
1,5	1,5	2,25	3	3,75	4,5	5,25	6	6,75	7,5
2	2	3	4	5	6	7	8	9	10
2,5	2,5	3,75	5	6,25	7,5	8,75	10	11,25	12,5
3	3	4,5	6	7,5	9	10,5	12	13,5	15
3,5	3,5	5,25	7	8,75	10,5	12,25	14	15,75	17,5
4	4	6	8	10	12	14	16	18	20
4,5	4,5	6,75	9	11,25	13,5	15,75	18	20,25	22,5
5	5	7,5	10	12,5	15	17,5	20	22,5	25

Figure 3.3: Assessing the Risk Potential

The result is a risk matrix showing the risk potential for all building blocks in the modeled smart grid environment. Depending on its value (i.e., probability level multiplied by impact level), the risk potential has been defined as low (green), medium (yellow) and high (red), see Fig. 3.3. This approach allowed us to identify potentially high risks in European smart grids. For high-risk domains it is advisable to identify the individual smart grid components causing the high risk potential, therefore, we are currently performing technical security audits (see Section 3.3).

Risk Mitigation

Based on the risks identified in phase II and assessed in phase III of our smart grid risk management approach, mitigation strategies are subsequently developed in phase IV. The goal of the mitigation strategies is to either decrease the probability of a successful attack, or to alleviate its impact, possibly also both at the same time. We are currently identifying suitable mitigation actions for the individual risks by addressing each of the 31 threats individually. For each threat, generic measures are first defined (such as introducing a Public Key Infrastructure to counter risks that emerge from insecure handling of cryptographic keys). Subsequently, specific measures for the individual architecture building blocks (see Section 3.2) are identified. We are focusing on mitigation actions suitable for establishing a basic level of protection in order to ensure a broad application among the utilities. Additionally, advanced controls for a higher security level are defined, which can be implemented by utilities with more mature security management processes.

Compliance Check

In order to maintain a high level of the overall smart grid system security, it is important to include automated security compliance checks. These checks should be run against all infrastructure components. Depending on the component type, the tool should check whether the device configuration (such as the firmware version or the currently deployed configuration file) adheres to the latest protection and mitigation strategies. If not, the tool can identify specific components that need to be updated accordingly. Since a single vulnerable component in the smart grid can compromise overall system security, it is highly important that all deployed system components are known to the automated checking tool. We thus advise utilities to include the tool setup into the regular deployment processes.

3.3 Evaluation and Discussion

The following section describes the findings we came up with when applying our five-step risk management approach together with distribution systems operators, and comments on the necessity of complementing the theoretical approach with practical security audits.

Risk Landscape

The risk management approach outined in Section 3.2 allowed us to identify areas (i.e. architecture components) in European smart grids which show high risk potential in terms of cyber attacks. Specifically, our analysis showed that there are significant risks in the Functional Buildings, Customer Premises and Grid Operation domains. Regarding centralized components such as the Grid Operation and SCADA system, the probability of a security breach is relatively low as an outside attacker typically has no physical access to these components. Moreover, protection mechanisms are not prone to cost pressure on this level. However, once an attacker manages to get access to these systems, the negative impact will eventually be high; for instance, shutting down a primary substation node could affect whole city districts.

In contrast, security breaches targeted on decentralized components, which are deployed typically in the Functional Buildings and Customer Premises domain, are much more likely as attackers can easily get hold of these components. Attacks on these components are facilitated by the fact that the Smart Grid Gateways are accessible via Internet, and a lack of software security or a misconfiguration may be easily exploited. While the probability of an attack is high, the impact is expected to be limited at first. This may however turn out wrong as soon as a successful smart meter mass attack is published on the Internet, potentially leading to unanticipated cascading effects in the power grid. Thus, not only the probability, but also the impact of a successful attack occurring within the so-called "last mile" are possibly high, which explains the high risk potential.
Our analysis showed that a general risk affecting most of the architecture domains is a lack of secure authentication methods. A potential consequence is that system components accept malicious data or control commands from unauthorized sources, which could have strong negative impacts on grid stability. We therefore recommend broad use of standardized authentication mechanisms such as digital certificates, role-based access control, and two-way authentication for remote maintenance access points.

Security Audits

For high risk domains, it is advisable to identify the individual smart grid components causing the high risk potential. For these components, individual technical security audits should be performed by independent auditors in order to assess the technical risks and their reasons. According to the risk potential, we suggest two types of security audits.
The first type of security audit is a typical network and lightweight software security audit. For the chosen smart grid component (i.e. a smart meter), it focuses on network and communication security. Similarly, the lightweight software security audit analyzes how the component's software implementation reacts on security test inputs such as maliciously modified network communication or test input generated through fuzz testing. However, the monitoring of the component's software is limited to the communication with the device. For instance, if a test case leads to an unexpected device response or a crash, a potential vulnerability is identified, but it is not further investigated due to the limited technical access on the device hard- and software internals. The audit is thus feasible with limited resources such as limited time or device access.
The second type of security audit is an in-depth hard- and software security audit starting at the point where the first audit type ends. The audit includes a low level hard- and software security analysis including hardware disassembly, physical port accesses as well as both static and dynamic software analysis. In comparison to the lightweight audit, this type of analysis is extremely powerful and can uncover a wide range of vulnerabilities. Besides, it is also possible to demonstrate proof-of-concept attacks and estimate the severity of these attacks on a larger scale. The drawback of the analysis type is the high effort with respect to analysis time and costs as well as the requirement of a dedicated test system that can be physically dissembled and possibly damaged in course of the analysis.
For instance, our analysis showed that smart meters have a high risk potential, mainly due to the easy physical accessibility by attackers as well as the severity of potential large-scale attacks. Due to the requirement of a testbed, we set out to create a security test system comprising a

smart meter, a PLC Data Concentrator as well as a Headend system. On this test system, we are currently performing light-weight analyses on the components. Due to the high risk potential, the smart meter is also subject to an in-depth hardware and software security audit. This allows us to get a spot sample of how secure these systems are currently, and to develop tailored mitigation strategies.

3.4 Conclusion and Future Work

We have presented an architecture-driven approach for smart grid risk management capable of bridging the gap between a high-level architectural view and specific technical security measures. Our approach cannot replace a risk analysis per se, as technical smart grid implementations and employed products differ significantly between users such as utilities or energy providers. It is, however, the first step in a utility-centric smart grid risk analysis that needs to include low-level technical implementation specifics as well, and may help users to identify areas with high risk potential, and to focus the mitigation actions on them.
Future smart grids will integrate a wide variety of different technologies. Therefore, the crucial challenges are to make sure that cybersecurity and interoperability requirements are satisfied. We argue that these issues can only be solved by a national smart grid reference architecture. Such a reference architecture would specify the minimum security requirements for the individual components and make sure that devices are carefully designed in accordance with them. At the same time, seamless interoperability would be ensured by defining appropriate interfaces. Individual implementations could still be derived from the reference architecture by instantiating specific domains.
Currently, there are no obligations for device vendors and utility providers to stick to the recommendations and guidelines published by existing standardization bodies. Therefore, a corresponding legal and regulatory framework should ensure that the minimum requirements defined by the reference architecture are followed. On the other hand, it must be ensured that the reference architecture is not only followed due to legal obligation, but rather broadly accepted by the different stakeholders. Therefore, all relevant stakeholders must be adequately involved in the process of establishing the reference architecture from the very beginning.

3.5 Acknowledgements

This work has been partly funded by the project $(SG)^2$ under national FFG grant number 836276 through the KIRAS security research program run by FFG and BMVIT.

CHAPTER 4

Physical Attacks on Smart Grid Devices

Devices for the smart grid are - compared to classic IT systems - special in the sense that they are often located in an area that is physically accessible to attackers even though they are considered to be highly security-critical. Examples are smart meters that are located in private households, or intelligent devices that are placed outside the trusted environment of the network operator's premises. Such devices are therefore prone to physical attacks, where the attacker is able to make use of more potent methods to compromise a device in comparison to the techniques that are applicable remotely (i.e. over a network connection). The current chapter describes extensively and provides examples of the different types of physical attacks that can be utilized to attack the networked embedded systems typically located in critical smart grid field components.

Having physical access to devices opens a completely new dimension of attack vectors. Target of such physical attacks can be e.g. smart meters, PLCs, local distribution network transformers, actuators, sensors, PV devices, gateways, and data concentrators. The embedded nature of such devices often gives a false impression of security: while for a mobile device, like a laptop, it is obvious that an unencrypted hard disc can be read out easily when an attacker gets his hands on the device, it is less obvious that this applies to many other forms of storage that appears in such equipment.

4.1 Goals of Physical Attacks in the Context of Smart Grid Devices

The goal of an attacker of a smart grid device can be classified into the two categories *Information Gathering* and *Manipulating the Device Under Attack* which are described in the following.

Information Gathering

Here the attacker gets illegitimate access to information. This attack scenario refers to the violation of the generic security objective *confidentiality*. Although much of the data in the smart grid is machine generated (such as sensor values), the impact on privacy of the users may still be high. The most prominent example is here the potential access to smart metering data, which led to significant public concerns against the application of smart metering devices. However, when it comes to physical attacks, the primary impact on privacy is usually limited. This is because the attacker can potentially access only information that is stored in the device, a device he was already able to get physical access to, either because it was in his premises or because he was able to illegally get access to it. The more significant target for an attacker is the acquisition of information that can be used in further steps for additional attacks: these can be network-based attacks and for manipulating the device under attack (see below). One important type of information here is key material: if a symmetric master key is stored on the device, that is common to a certain class of devices, this key can be used to compromise a huge number of devices remotely. Individual symmetric keys, and private keys of asymmetric cryptography are usually less problematic, but they still allow an attacker to impersonate the device and use e.g. a trusted connection to bypass firewalls, and perform further attacks on the other endpoint of the connection. But not only passwords and keys are interesting for an attacker, by extracting and analyzing the firmware and software that is running on a device, an attacker can identify vulnerabilities or hidden device features in the code that can be used for standard attacks such as buffer overflows.

Example 4.1

During a local attack on a smart metering device, an attacker discovers a proprietary engineering command in the firmware allowing the power switch to be controlled. The attacker transmits this command to other smart meters on the network in order to blackout surrounding households.

Finally, the extraction of software or firmware is relevant with respect to intellectual property (IP) protection. Intelligent field devices in the smart grid often rely on algorithms, parameters, and other form of intellectual property. While these attacks do not impair the stability of the smart grid, they may introduce severe financial harm to those producers that had large development costs for their products.

Example 4.2

During a local attack on a smart metering device, the attacker gets access to a master password, which allows him to access a restricted part of the smart meter's web-interface where metering parameters can be configured. With that information, any smart meter can be manipulated to record less consumption without breaking any seal.

Manipulating the Device Under Attack

Here the attacker manipulates the functionality of the embedded device in an unauthorized fashion. This refers to the violation of the generic security objectives *integrity* and *authenticity*. Manipulation of the mechanical and electrical parts of a smart grid device is always feasible if an attacker gets physical access to the device, and is not a matter of IT security. However, being able to manipulate the programmed functionality of a smart grid device can be a critical threat to the smart grid, since such an attack can be simultaneously applied to a large number of devices by a single attacker. A manipulated device can also be used by an attacker as a platform to compromise other parts of the system, since network-based security measures usually prevent attacks from outside. A device that is compromised by a physical attack might have privileged access to other components in its network segment, and therefore act as a beachhead to attacking further devices.

Example 4.3

An intelligent device that is used to control a circuit breaker is part of a control system of the substation it belongs to. The substation is together with other substations connected to a central SCADA system, which is usually a PC-based platform. If an attacker can successfully compromise a single circuit breaker's control system, it could use the field bus system interconnecting the sensors and actuators in the substation to send commands to these sensors. In current technology, there is are usually no authentication and integrity measures implemented at that level, such that doing so does not involve any further weaknesses of the system. In addition, the attacker can use the uplink connection to compromise the SCADA system. Chances are much better here, since the attacker does not have to overcome any firewall and other isolation measures, and the system is probably more vulnerable on this interface, as attacks on this side are mostly not expected.

Attack potential	Techniques	Typical equipment	Typical equipment costs (EUR)
Low	Access to local storage Accessing open interfaces Probing on buses Simple faults	memory chip reader, logic analyzer, microcontroller/FPGA boards	less than 1,000
Medium	Simple side channel attacks Glitching Attacks	digital oscilloscope, signal generator, FPGA boards	2,000 - 10,000
Elevated	Enhanced side channel attacks EMA DPA Template attacks Semi-invasive attacks	high-resolution digital oscilloscope, FPGA boards, chemical depackaging	10,000 - 50,000
High	Invasive attacks Fault Attacks	(laser) probing station, chemical depackaging, focused ion beam (FIB)	more than 50,000

4.2 Overview of Physical Attacks

Physical attacks [123] are very powerful in general, but sophisticated attacks require not only expert knowledge but also increasing resources in terms of time and money. The following table gives a classification of different techniques with respect to their attack potential. It is important to note that especially the classes of attacks with low and medium attack potential are relevant candidates in the context of smart grid security.
The techniques with low attack potential in this table are different from the others in the sense that they typically target systems without security functions like encryption and integrity checks. The attacks of the other three classes on the other hand can target weaknesses in already established cryptographic routines as well.
Figure 4.1 shows a typical taxonomy of physical attacks. In general, there are three categories depending on how invasive (i.e. non-invasive, semi-invasive and fully invasive) the attacks are with respect to opening up integrated circuit (IC) microchips inside an embedded system. With non-invasive attacks, the ICs are not opened at all. The attacks are thus mostly limited to the electrical signals accessible from the printed circuit board (PCB) within the system as well as possible device emanations that can be used for performing side channel attacks. Non-invasive attacks can be very powerful already as the attacker has full control over the system environment (such as electrical signals, temperature, system clock, etc.). Semi-invasive attacks go further by opening up ICs so that the die within is visible (either from the front or from the backside). The IC stays fully functional as the isolating and protecting passivation layer is not removed. This allows the attacker to see what is inside the microchip and thus also perform optical attacks such optical fault injection or optical emanation analysis. With invasive attacks, the protecting

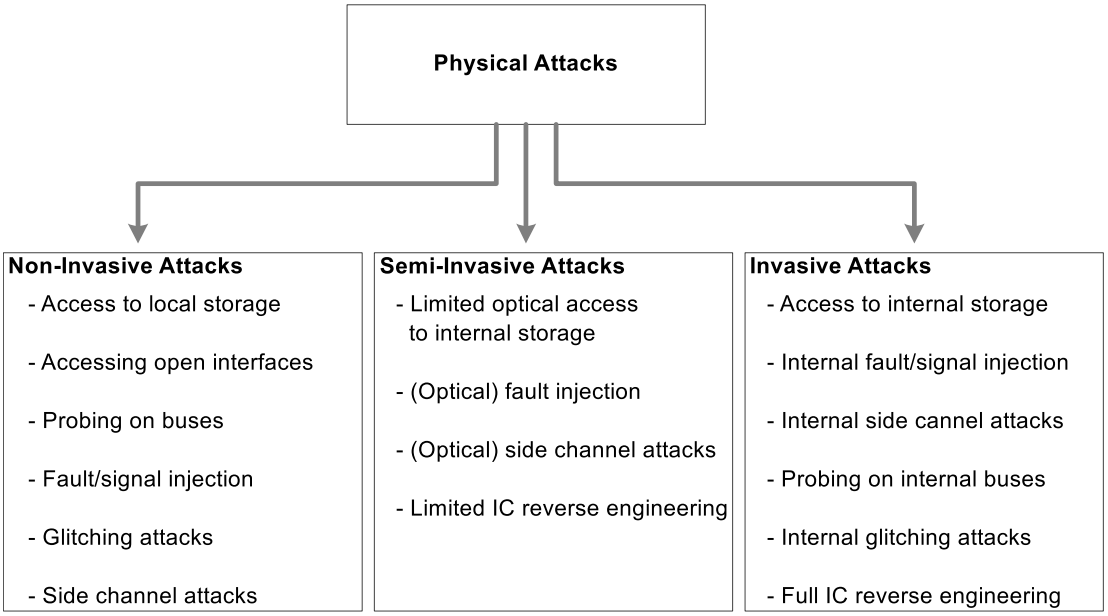

Figure 4.1: Taxonomy of physical attacks

passivation layer is removed as well. If the IC is kept functional, this allows the attacker to observe communication on internal buses or even inject own signals for the attackers advantage. On the other side, invasive attacks also allow full reverse engineering of microchip internals. In the following sections, different attacks within this taxonomy are described in more detail.

Access to Local Storage (Non-Invasive)

Accessing the information stored on an embedded device is often very easy, if information on these devices is not encrypted. Some devices are equipped even with removable and/or highly standardized storage such as SD cards, memory cards, hard discs, etc. But also any other form of non-volatile memory is not protected by its embedded nature. Memory chips such as flash or ROM can be unsoldered and read out by memory chip readers. For more complex settings, e.g. when no standard modules are used, the unsoldered memory chip can be integrated on a microcontroller board together with a self-written read-out routine. Bus probing (see below) is hereby helpful to reverse-engineer the protocol between the memory module and the processor. Figure 4.2 shows how this low-cost attack can be conducted on a typical smart meter device. The flash memory was removed from the smart meter (left) and then re-soldered to a breakout board (right). As the breakout board makes all flash memory chip device pins available through its connectors, a microcontroller board can be easily connected to read out the full memory content (i.e. the firmware) of the smart meter by utilizing a self-written routine. If required by the attack, data and code on the memory modules can be also altered, or the memory module may be replaced at all by the attacker's own memory module in the original device.

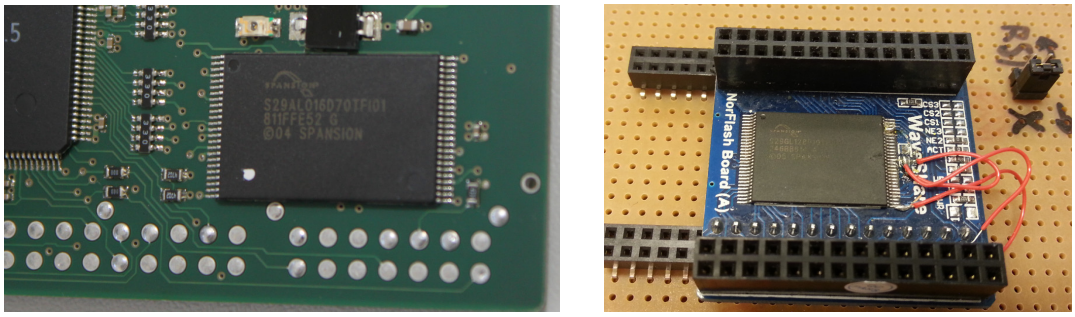

Figure 4.2: Flash memory chip in a smart meter (left), smart meter memory chip soldered to breakout board for readout

Accessing Open Interfaces (Non-Invasive)

Embedded devices come with plenty of interfaces. Network interfaces are always good attack targets. Smart grid devices, such as PLCs or PV control systems are usually not supposed to be connected directly to the Internet. Therefore one might find weaknesses in the network stack of wired network interfaces that are already patched for most standard IT systems. Ethernet with TCP/IP is used for those network connections, but there are other standards such as power-line communication or serial protocols, including MBUS. Power-line suffers also from the fact that many consider the physical nature of the interface, i.e., being modulated on the power line, as a sufficient barrier for an attacker. In fact, even with simple equipment it is not. In addition, the local attacker can also access wireless network interfaces. Beside communication based on IEEE 802.11, and the well-known weaknesses in WEP and WPA [128], a smart grid device might support other protocols such as GPRS, Wireless M-Bus, or even protocols from the home automation field (ZigBee, 6LoWPAN, etc.). A very important class of interfaces for local attacks comprises debugging ports, such as JTAG or serial consoles. These debugging interfaces allow an attacker to get intra-chip information during runtime with little effort and without depackaging the chip.

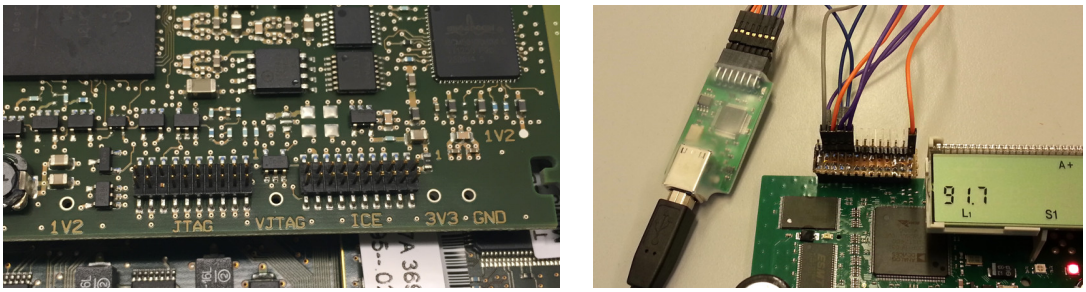

Figure 4.3: JTAG and In-Circuit Emulation (ICE) ports inside a substation automation system (left), connected JTAG interface on a smart meter (right)

Example 4.4

The JTAG debugging port can be used to read out any register in a chip in operation. Depending on the chip, the Test Mode Select Input pin has to be activated, and then the registers can be read out bitwise. Any key that is stored at some time in a register - and this is the case if any cryptographic operation is performed by the IC - can be read out by this method. Similarly, the full firmware can be read out from non-volatile memories just the same way.

Bus Probing (Non-invasive)

An alternative approach to gather information from embedded devices is to listen to the information that is exchanged on the platforms bus systems. A simple logic analyzer is hereby sufficient. Logic analyzers that can be connected with USB to any computer are available already for a few Euros.

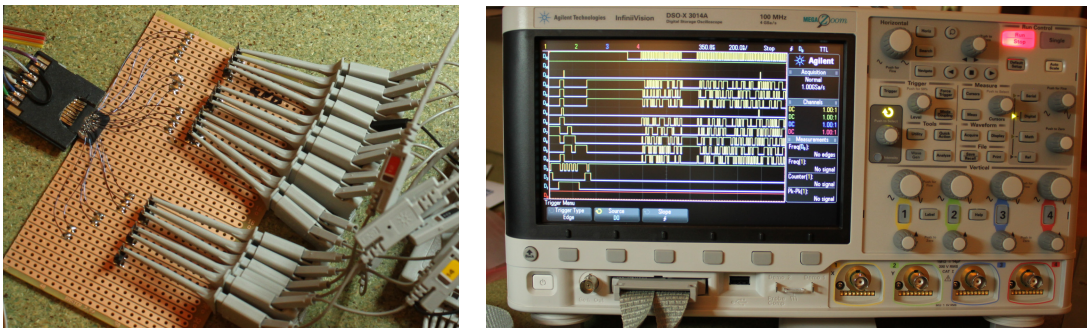

Figure 4.4: Bus probing on the internal bus of a microSD card

Figure 4.4 shows that bus probing attacks are even feasible in case microscopic circuit board traces. On the left side a very thin wire has been soldered to the exposed circuit traces on a microSD card that interconnect the internal card controller with the flash memory chip. To connect the logic probes to those wires, a simple breakout board was used. On the right side the intercepted signals are visible on the logic analyzer.

Fault/Signal Injection (Non-Invasive)

In contrast to non-invasive bus probing attacks where the attacker passively intercepts the signals on a bus, it is also possible to take an active role by purposely modifying or even injecting own signals. This can be done rather easily by utilizing readily available microcontroller or FPGA development boards. Assuming the simple case that inside an embedded system two devices communicate with each other, there is usually a sender and a receiver. Unless more advanced techniques such as bus arbitration are used, these roles between the devices are typically fixed.

However, as signal injection is done by a second sender (i.e. an FPGA board) this could lead to issues as both senders would drive the bus in different directions. The solution is to perform a man-in-the-middle (MITM) attack where the connection between the two original devices is cut and the FPGA board is inserted in between. This allows the attacker to selectively forward and arbitrarily modify any communication between the two original devices.

Example 4.5

A PV smart grid device communicating over an unencrypted but proprietary PLC (power line communication) protocol has a SoC controller chip and a modem chip on its circuit board. By probing the bus between the controller and the modem chip, the attacker was able to identify the messages that report how much power is fed into the utility grid. As the PLC protocol behind the modem is unknown to the attacker, he uses a cheap microcontroller board to modify the power measurement value transmitted on the bus to the modem chip. The modem chip accepts the modified message and sends it over to PLC network to the utility. Ultimately, this allows the attacker to report an arbitrary amount of generated power.

Glitching Attacks

Most common CPUs, FPGAs or microcontrollers are based on synchronous logic meaning that they require a system clock signal. Each time the clock signal occurs, the internal logic of the device advances to the next state. For instance with a CPU, this could be an instruction that is executed. Besides, the device needs to be powered by supplying a voltage to its power pins.

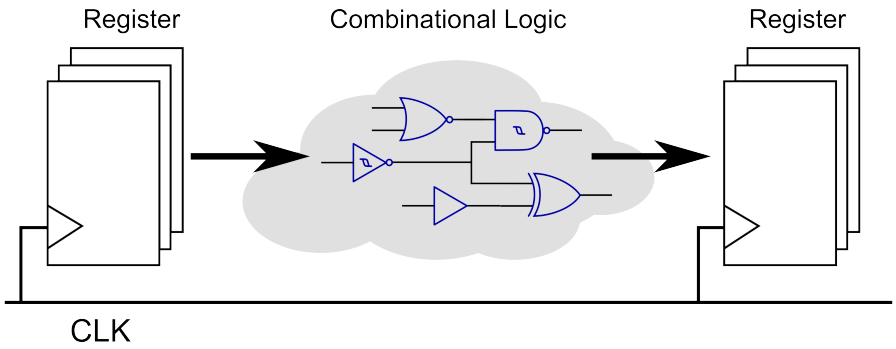

Figure 4.5: Synchronous Register-Transfer-Logic (RTL)

Figure 4.5 shows a greatly simplified version of how a synchronous system typically works. On the left side, the current computation result is stored inside a register (i.e. a set of Flip-Flops). If the clock signals the system to continue, the register makes its internal state available on the output. Consequently, the data is transferred through the combinational logic block and, after

enough time has passed, the result of the computation is available at the input of the subsequent register on the right. The time it takes for all signals to go entirely through the combinational logic block typically accounts for the major part of the required delay between two adjacent clock events (i.e. the maximum possible clock frequency of the system). Real-world systems comprise a huge number of such Register Transfer Logic (RTL) blocks ultimately triggered by the system clock. With glitching attacks [101], several of the physical properties of a system can be exploited to perturb the operation of the system to the attacker's advantage. The two predominant types of glitching are *clock glitching* and *voltage glitching*.

Clock Glitching

With clock glitching, the attacker supplies the clock signal to the system just as within a usual system. However, at a time of the attacker's choosing (for instance when a security critical instruction is executed), one or more intentionally too fast clock pulses are supplied. If the delay between those clock cycles is less than the time required for the data to pass through the combinational logic block, the input to the next register does not contain the finished computation result. Instead, some of the signals are still in their previous or in an intermediate state while others might be finished already. Besides, not all RTL blocks have the same time requirements. Considering a CPU logic implementation, the combinational logic within a complex CPU instruction might have a significantly higher logic delay in comparison to comparably simple implementations such as the CPUs program counter. As a result, clock glitching is especially effective against security critical conditional jumps or cryptographic computations in the firmware.

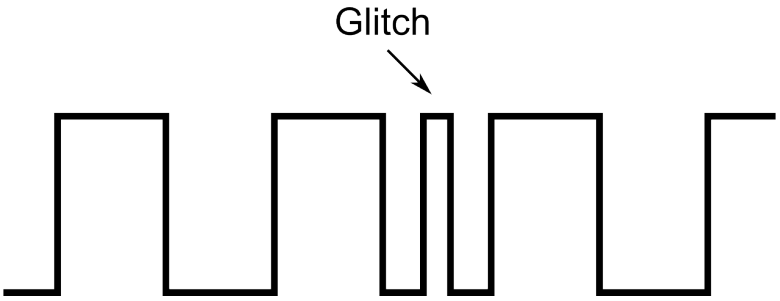

Figure 4.6: Glitch in the clock signal

Figure 4.6 provides an example of how an idealized clock glitch signal could look like. While clock glitching attacks can be conducted with lab equipment such as signal or pattern generators, the low-cost approach typically uses FPGA boards for clock and glitch generation. In addition to finding the right glitch parameters (i.e. number of glitches or glitch duration), the key question is when to start the glitch attack. While more advanced techniques are available, brute force (i.e. trial and error) based approaches often work well if the test setup is automated. For instance, this can be achieved by automating the FPGA glitch generation and system interaction with custom software or scripts running on a PC.

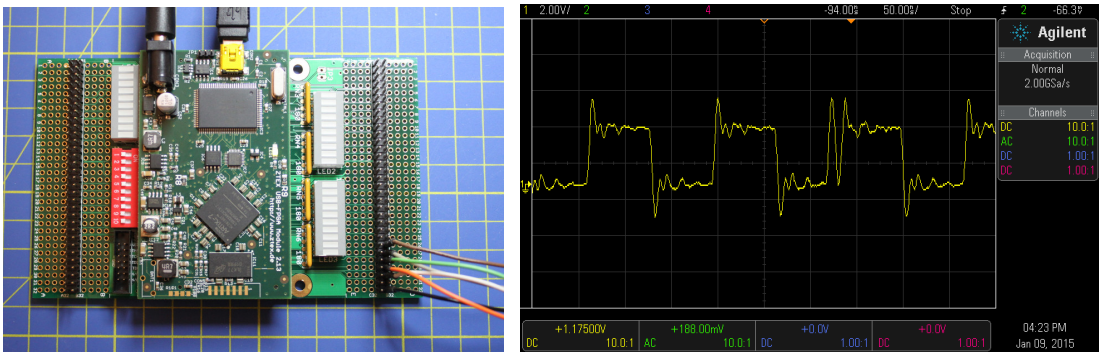

Figure 4.7: FPGA development board used for clock glitching (left), generated clock glitch measured at the target device (right)

Example 4.6

A charging station for electric vehicles at the customer's premises is connected to a secure communication network. The attacker discovers that the device can be switched into a service mode if the utilities service password is entered. Since the attacker doesn't know the service password, clock glitching is applied to transform the execution of the conditional password checking branch instruction in the system's CPU into a non-branch instruction. The result is that although an invalid password was entered, the attacker can "jump over" the password check and enter the service mode. The attacker discovers that through the service mode, the credentials and encryption keys of the utility's secure communication network can be read out.

Voltage Glitching

A change of the supply voltage impacts the device operation in multiple ways. If a lower voltage is used, the overall logic delays increase. Within an integrated circuit, signal traces have a capacitance that is for instance impacted by the length of the trace. Each time the state of the signal changes (i.e. from a logic 1 to a 0 or vice-versa), the driving transistors need to transfer current until the desired new state has been achieved. The less voltage is available for this task, the longer it takes. Hence the maximum operating frequency of the IC is decreased as well which can render the IC more susceptible to clock glitching attacks. Besides combinational logic, different types of memory such as flip-flops, registers and various types of RAM and non-volatile memories are affected through the voltage change as well. Memories often work by comparing a stored charge with a threshold reference voltage. Depending on the memory technology, the memory content could be interpreted as logic 1 if the stored charge is higher than the provided threshold. Otherwise, the state is considered to be a logic 0. However, if an attacker changes the threshold voltage far enough, the stored charge in a memory cell could be interpreted the wrong way. Since the overall device operation is impacted by the voltage change, attackers typically

either increase or drop the supply voltage to a device only for a short period of time. The result is a sudden voltage glitch causing memories such as the registers in between combination logic to output an invalid state. Similar to clock glitching attacks, this can be effectively used to bypass security checks or perturb cryptographic computations. Figure 4.6 shows an exemplary voltage glitching setup. The FPGA board on the left as well as the target system under attack is controlled by custom software on a PC. A resulting voltage glitch waveform is depicted on the right.

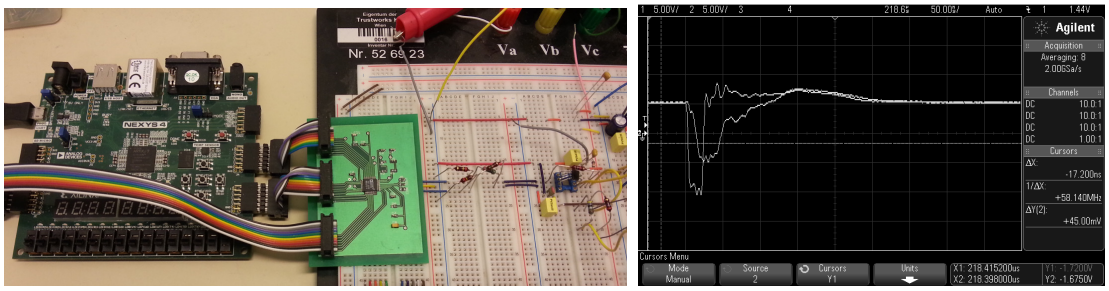

Figure 4.8: FPGA based voltage glitching setup (left), voltage glitch waveform and supply voltage measurement (right)

Side-Channel Attacks

Side-channel attacks, first introduced by Kocher [63], exploit the implementations of cryptographic algorithms or software. When performing a side-channel attack, some observable behavior of the (cryptographic) routine implementation is used to obtain additional information that allows the attacker to decode some cipher text, calculate the cryptographic keys or obtain details of the executed instructions and data within the system. This is in contrast to classic cryptanalysis, where weaknesses of the cryptographic primitive itself are exploited. Side-channel attacks can be classified along two axes (Fan et al. 2010):

1. Invasive vs. non-invasive: Invasive attacks require opening the device under attack. This usually refers to the chip level, where depackaging of the chip might be needed. Invasive attacks can be further divided into semi-invasive and fully-invasive attacks. The difference is that with semi-invasive attacks the passivation layer of the chip stays intact whereas with fully invasive attacks, the chip is further deprocessed depending on the requirements of the particular attack. In the context of the smart grid, also another abstraction level is relevant, namely whether the embedded device needs to be opened (and therefore seals are broken). However, for most side-channel attacks this is case.

2. Active vs. passive: While passive attacks restrict themselves to only observe the device's behavior, an active attack also manipulates the device's operation e.g. by injecting various types of faults (electrical, optical, etc.) or by employing glitching attacks.

Since side-channel attacks base on physical phenomena and not (only) on mathematics, there are numerous attacks and one can be sure that for the future there will be even more. The most common attacks are, in increasing order of complexity:

- Timing attacks
- Power analysis attacks (SPA, DPA, Template attacks)
- EM-attacks

Side-channel attacks can be very sophisticated, and, as shown in Table 1, also very expensive. Except for smart metering, where e.g. by the German BSI the usage of secure processors is enforced [36, 37], field devices in the smart grid usually come with little security functionality. Therefore, side-channel attacks might be considered less relevant compared to the simple attack methods presented above.

Timing Attacks

Timing attacks [63] exploit data-dependent execution time differences. Consider the password check illustrated in Listing 4.1. The password check uses the user-supplied password passwd and compares it against a stored one. At the first glance the password checking routine seems to be secure if a long enough password is used to thwart password guessing attacks. However, a close look into the code reveals that the password immediately returns "false" as soon as the first character in the supplied password is different from the stored one. Instead of a conventional brute force password guessing attack, an attacker can thus measure the time between sending the password guess to the system and the response that the supplied password was wrong. However, if the first character was correct, this response will come back to the attacker a short period of time later since the loop in the password check is executed once again. Due to this timing information, the correct password can be easily guessed in comparison to a conventional brute force attack. However, as the approach needs reliable timing measurements averaging steps are often necessary so that over a higher number of measurements potential non-data-dependent timing differences can be filtered. In fact, this approach works so well that vulnerable cache-timing software implementations can be even attacked over multiple hops on the Internet [12].

```
1 bool check_password(char *passwd)
2 {
3    for (int i=0; i<pass_len; i++)
4    {
5       if (passwd[i] != stored_passwd[i])
6          return false;
7    }
8    return true;
9 }
```

Listing 4.1: Password check implementation that is prone to a timing attack.

Figure 4.9: Optical interface connected to a smart meter for optical timing analysis testing

Example 4.7

Smart meters typically have an optical port allowing service and configuration settings to be changed. Since many values can neither be read nor written without the utility password, an attacker uses a low-cost optical interface and measures the response timing for password guesses. Using these attacks, s/he discovers that the password check is vulnerable to timing attacks and the password can thus be easily guessed by the attacker.

Power Analysis Attacks

Whenever synchronous logic receives a clock signal, the output of a register is sent through combinational logic and finally reaches the next register (register transfer logic). During that time, the transistors in between need to switch so that single signals or a whole bus get switched from one state into the other. This does not only take time, but it also draws current because of the switching and the capacitances of the various structures within the chip. If the power consumption is measured over time, the resulting power trace is different and thus characteristic for each logic block. If for instance the power consumption of a CPU is observed between clock cycles, each executed individual instruction will have a different power trace. The reason is that within the microchip implementation, the logic that implements the CPU instruction is different as well.

Simple Power Analysis (SPA): The main idea of simple power analysis [64] is to directly analyze the power trace of a microchip during security relevant tasks. As each operation has its individual power signature, it is possible to determine which operations are performed within the chip by solely looking at its power consumption between clock cycles. In fact, the power trace of the execution of the same operation looks slightly different if the data supplied to that operation is different as well. The reason is that the more internal states need to be switched, the higher is

the power consumption at a specific time. For systems which are not especially hardened against these kinds of attacks, an attacker might thus be able to determine which operations are performed within a microchip by solely looking at the power trace. Even more, the attacker might be able to determine the data that is processed by these operations as well. The impact of these attacks can be especially severe if security critical information such as key material or credentials can be extracted this way.

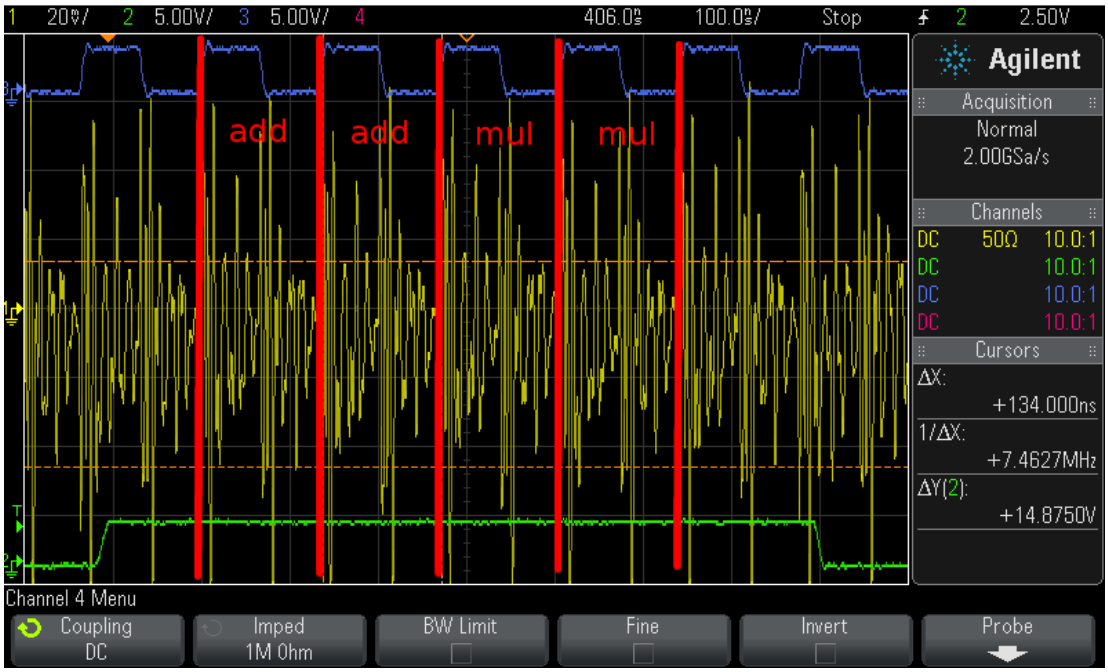

Figure 4.10: Exemplary power trace of a microcontroller during the execution of a simple algorithm

Figure 4.10 gives an example for a simple power analysis (SPA) attack. In the top of the picture the system clock signal is visible while at the bottom, a custom trigger signal for the measurement has been inserted. The power trace is visible in the center of the picture (yellow). As visible, the two multiplication instructions executed have a different power trace as the first two addition instructions. With filtering setups and the use of averaging over a high number of equal test runs, the attacker can get a better signal to noise ratio by canceling out measurement noise. The more precise the power measurements are the better the executed operations and the processed data can be identified. Mainly depending on the clock frequency the attacker requires a reasonable priced digital oscilloscope to measure power traces and conduct a simple power analysis attack. The power is usually measured over a small shunt resistor between the power supply and the device power pin.

Differential Power Analysis (DPA): Simple power analysis attacks usually involve significant manual analysis effort and are easy to protect against if the power consumption is internally filtered or randomized (i.e. through the insertion of dummy cycles). Differential power analysis attacks

are much more powerful. Here, an attacker takes a high number of power trace measurements first. In the next step, a power model (i.e. hamming weight) is used to compute the theoretic power consumption of a (cryptographic) algorithm with a small number of guessed bits. For instance, in the case of the AES encryption algorithm this could be the first key byte that is used in the first AES round during AES computation. For this first computation, the attacker thus has a high number of measured power traces from the target device as well as the idealized and theoretic power consumption for the key guess using the power model. In order to determine whether the guess was correct, the attacker uses statistical means (i.e. the statistical correlation) to determine "how good" the match between the theoretic power consumption and the real measurements is. This is done for all key guesses (i.e. for 255 key guesses when the first byte of the AES key is targeted). If enough good measurements have been acquired, only one of them will show a strong statistic match. This is continued for the other key bytes as well until the full AES encryption key is recovered.

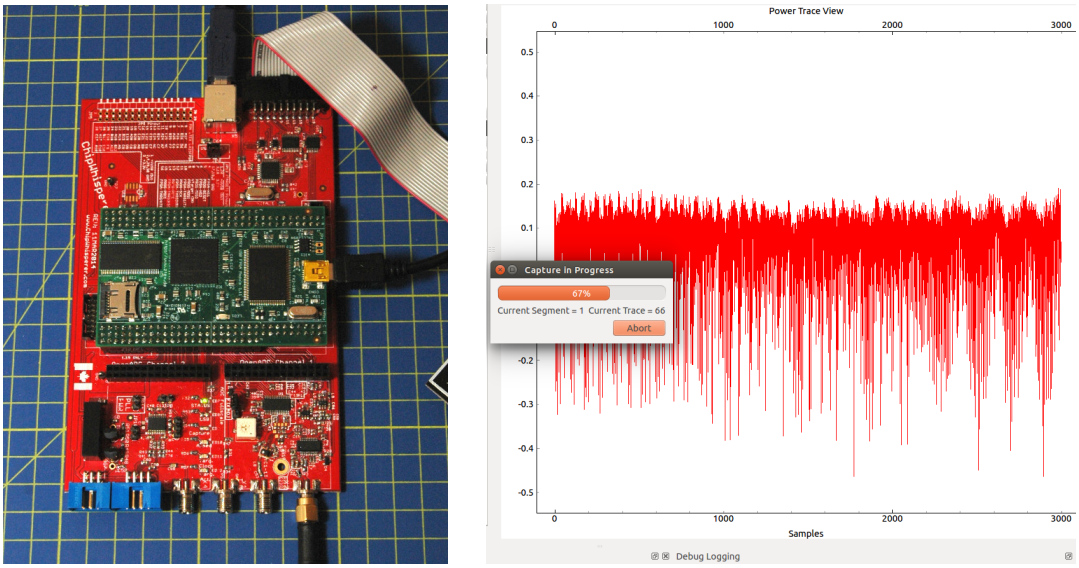

Figure 4.11: FPGA based DPA measurement setup

Due to the high number of measurements and the strong statistical methods, this approach can still lead to results if power analysis attack counter measures have been implemented. To take a high number of measurements with considerable sample length, once again an FPGA board based approach can be utilized (cf. Figure 4.11).

Template and Other Profiling Attacks: Similar to SPA and DPA, template attacks [19] use the side-channel information that is leaked through power consumption of cryptographic algorithm implementations. In contrast to the previous two approaches, it assumes that the adversary has access to an identical device, which is used to build a multivariate stochastical model of the signal and noise of the power trace. This model is the used to iteratively perform a maximum likelihood classification of a prefix of the power sample. Originally shown to be very effective for

symmetric cryptographic operations, also asymmetric cryptography has been shown vulnerable to template attacks [74]. The approach can be generalized to any other model-building technique. For instance, machine learning can be used to build a profile of the cryptographic algorithm which is then applied to solve the classification problem in the foresaid iteration [43].

EM Attacks

A very powerful method for side-channel attacks makes use of the EM emissions that arise from the data-dependent current flows inside a device [1]. To obtain sufficient information for a successful side-channel attack, a single probe can be sufficient. With multiple probes, more sophisticated attacks can be performed.

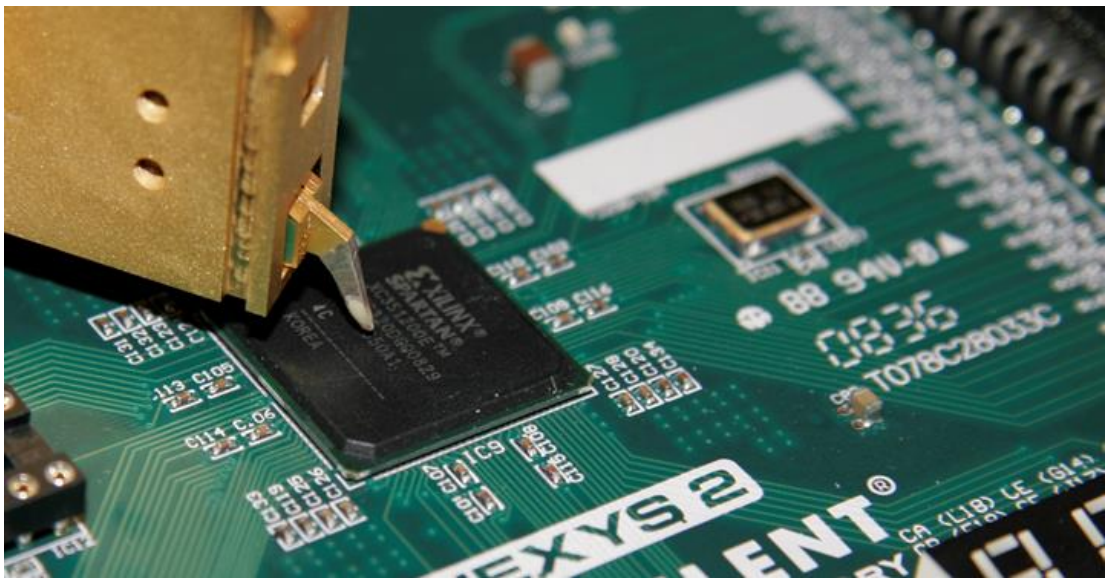

Figure 4.12: Setup for an EM attack (Image: Fraunhofer AISEC / Andreas Heddergott)

With EM attacks it is possible to observe much faster signals than with e.g. power measurements, since a larger frequency band can be recorded. For power measurements, the signal is often lowpass-filtered. The higher-frequency parts of the signal allows in particular capturing the effects of the combinatorial logic that is between the latches of the circuit. The results of EM attacks can be improved by opening the packaging of the device (see below). In analogy to power side-channel attacks, simple attacks with a single sample (SEMA), differential attacks with multiple samples (DEMA), and template-based attacks are possible.

IC Decapsulation

For semi-invasive and invasive attacks on Integrated Circuits (ICs) it is necessary to open up the IC device and expose the contained silicon die. Depending on the type of attack, the microchip should still be functional after the decapsulation procedure. As a preparation for these kinds

of attacks, a common approach is presented. However, other decapsulation and preparation techniques exist as well.

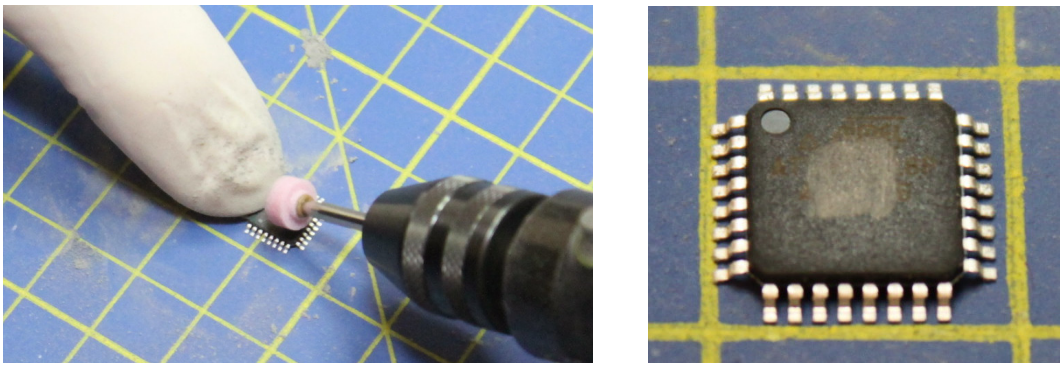

Figure 4.13: Milling a cavity into the desoldered chip

Initially, the microchip to be decapsulated is usually desoldered from the circuit board for easier handling. In the next step a cavity is carefully milled into the center of the IC package with a Dremel tool (Figure 4.13). The cavity needs to be deep enough to hold a drop of acid in place, but the die below has to stay undamaged from the milling process.

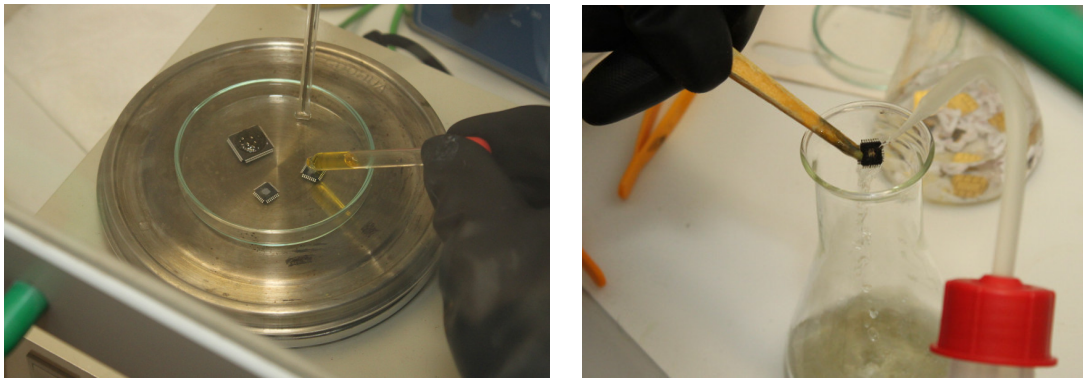

Figure 4.14: Chip decapsulation with nitric acid (left), rinsing with acetone (right)

In the next step, the chip is heated up and a drop of acid is carefully applied onto the milled cavity. Usually, nitric acid or sulphuric acid is used for this process (Figure 4.14). After the reaction of the acid with the epoxy package is finished, the chip is rinsed in acetone. The etch steps and rinse steps are repeated until the die is exposed. Depending on the type of attack, it is also possible to completely remove the package this way. For semi-invasive attacks the chip needs to stay functional and just the top epoxy cover of the chip is removed. As soon as enough material from the IC epoxy package has been removed, the chip needs to be cleaned for microscopic analysis (Figure 4.15). This is usually done in an acetone bath inside an ultrasonic cleaner. The result of the decapsulated and cleaned chip is visible on the right side of Figure 4.15. In this state the chip is still functional and the bonding wires are intact.

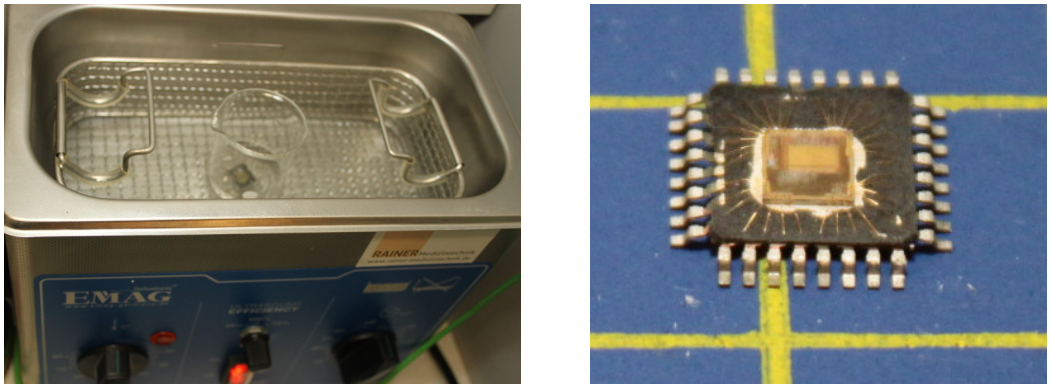

Figure 4.15: Chip cleaning in an ultrasonic cleaner (left), still functional chip prepared for semi-invasive attacks

Limited Optical Access to Internal Storage

Depending on the non-volatile memory type in a chip (i.e. mask ROM) it is possible that the content of the memory can be optically read either from the front or from the backside without having to deprocess the chip. However, the approach is limited to memory types that can be optically read without preprocessing the chip.

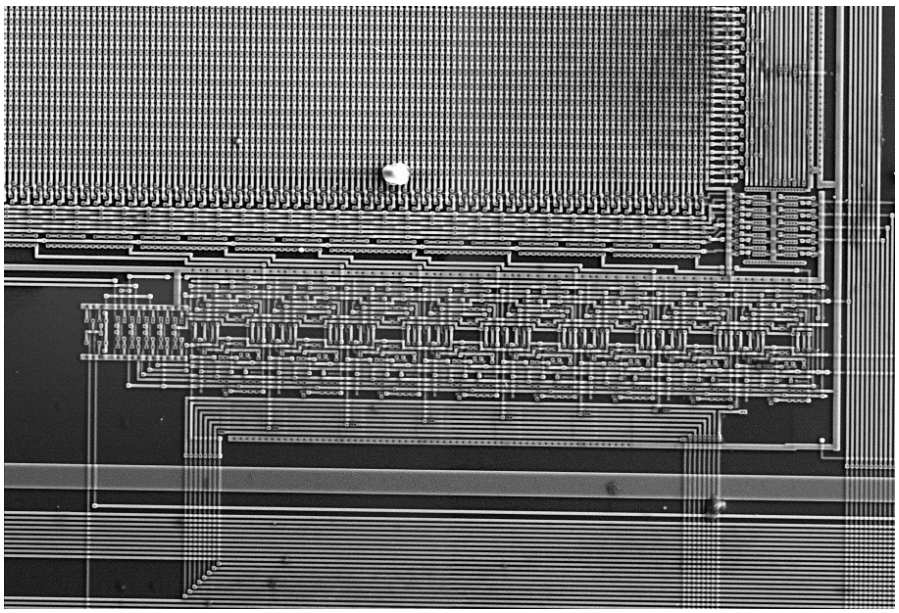

Figure 4.16: Lower part of a via-ROM memory and its column driver

Figure 4.16 shows a Scanning Electron Microscope (SEM) picture of a via-ROM memory. The memory content is set with tiny via plugs that can be seen optically. An attacker could thus gain access to the internal ROM storage content through a semi-invasive attack.

Example 4.8

An internetworked smart grid device in a secondary substation uses a proprietary encryption algorithm to secure the protocol exchanged over a wireless transmission link. The attacker was able to intercept the wireless traffic with a low-cost Software Defined Radio (SDR), but is unable to decrypt the traffic. He manages to acquire an outdated similar device through an Internet auction and decapsulates the encryption chip in it. He discovers that the chip uses a known CPU architecture and the firmware is stored in an optically readable mask-ROM memory. Using a microscope from a nearby university lab, he manages to dump the memory and reverse engineer the proprietary encryption algorithm. It turns out that the algorithm is weak and, therefore, the attacker manages to wirelessly control the secondary substation devices in his vicinity.

Semi-Invasive Fault Injection

Introducing faults into cryptographic routines can lead to leakage of information that allows computing secret key material [11]. Faults can be injected e.g. with a laser probing station, as shown in Figure 4.17. For this, the package has to be opened, as described in the previous sections. The wavelength of the laser is a limiting factor for the spot size, and therefore the area where the laser energy leads to bit-flips. On the other hand, the optical properties of the silicon have to be taken into consideration. This can be used to attack the chip from the backside with a laser that emanates in the IR range - for these wavelengths, the silicon die is transparent, and the gates are not covered with the metal layer, as it is the case when attacking the chip from the front side.

Figure 4.17: Fault attack with an optical laser (Image: Fraunhofer AISEC / Andreas Heddergott)

IC Reverse Engineering

Similar to the semi-invasive limited access to local storage, it is also possible to perform limited IC reverse engineering with semi-invasive attacks. If through the front- or backside of the IC enough interesting details about the implementation are visible, an attacker might be able to obtain security critical information. Figure 4.18 shows an example about how relevant parts of a logic implementation can be obtained from a cryptographic chip through semi-invasive reverse engineering.

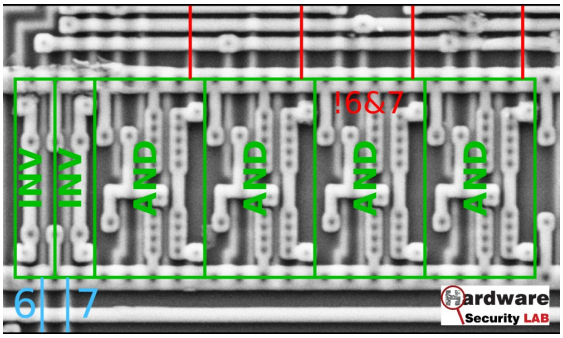

Figure 4.18: Obtained logic information from a semi-invasive attack on a cryptographic chip (SEM image)

If device secrets are deeply hidden in the silicon, fully invasive IC reverse engineering utilizes IC deprocessing and microscopy techniques to take apart the microchip implementation layer by layer. A typical CMOS IC has a poly-silicon logic layer at the bottom and several metal layers (Cu, Al) at the top which are interconnected with via layers (W) and insulation layers (mostly SiO2) in between. The logic layer at the bottom contains the actual circuit implementation with common design elements such as different types of memory, cryptographic cores, a CPU, peripherals or glue logic. A major challenge for deep silicon security analysis is the deprocessing of ICs with results that are suitable for image analysis. Typical approaches use chemical-mechanical polishing (CMP), wet chemical etching or plasma etching (Figure 4.19).

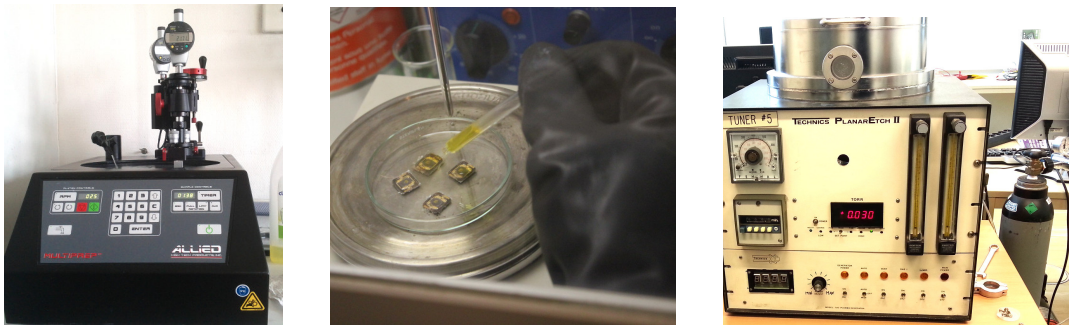

Figure 4.19: Polishing machine (left), chemical deprocessing (middle), plasma etcher (right)

By using these deprocessing techniques, the attacker can obtain internal information deeply hidden in the chip such as the implementation of proprietary encryption algorithms, the firmware contained within ROM memory (similar to the limited semi-invasive optical access to internal storage but more powerful) or potential counter measures in the chip.

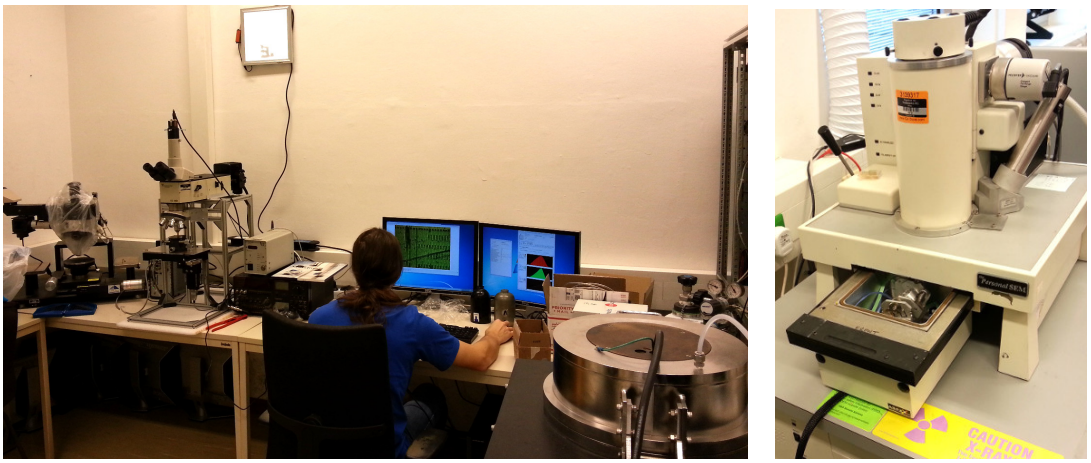

Figure 4.20: Microchip image analysis (left), scanning electron microscope (right)

4.3 Basic Protection Mechanisms

The simplest threats of local attackers can be countered by being aware of the possibilities a local attacker has, and choosing a system design that mitigates these threats. The first step is to be aware of all open local interfaces, and restrict the access to the system by these interfaces in the same way as one would do for a network interface that is potentially accessible to an attacker. The interfaces should obviously not allow any unauthorized access, state-of-art authentication mechanisms should be used, and the authentication credentials should be hard to guess. Functionality that is needs not be accessed on those interfaces should not be exposed. The possibility of standard attacks, like buffer overflows should be considered also for those interfaces. Vulnerabilities should be patched by updates. This might be a difficult task for embedded devices once they have been rolled-out. Not needed interfaces should be removed ideally not only from the casing but also from the circuit board. Probing on buses and signals can be impeded by obfuscating the data that is transmitted over them. Already simple methods, that are not cryptographically secure, can be very effective here, like scrambling the data according to a schema that is given by some shared pseudo-random generator. Of course, using strong cryptography would be even better here, but maybe not feasible e.g. on the lines between a processor and the system's RAM. Key material should be stored in secure places. Removable storage is the easiest to read out for an attacker, but from any kind of non-volatile memory it is possible to extract data. Secure memory is the preferred choice here. If it is not possible to integrate a secure memory or security processor in the device, sometimes Physically Unclonable Functions (PUFs) might be a possibility to securely store a key with which the key material is encrypted before being stored in NV memory.

Furthermore, also non-IT solutions to protect a device are feasible. Sealed cases are effective if breaking the seal can be related to a single person and legally prosecuted. However, one needs to think about potential goals of an attacker: somebody planning a terrorist attack is probably not be scared off by seals.

4.4 Conclusion

In this chapter, we provided an overview of physical attacks including example of how these could be used by attackers to attack the embedded systems in smart grid field components. While these attacks can be applied to unprotected embedded systems in general, it is important that critical smart grid devices need to consider these attacks in their design and they should rely on state-of-the-art secure hardware components that already include countermeasures against these attacks. We presented a taxonomy for physical attacks and showed that many of those powerful attacks can be carried practically and easily carried out by attackers once they have physical access to the embedded devices. By raising the awareness for these attacks and choosing mitigation strategies in the system design, we believe that critical infrastructure systems can be better protected in the future.

CHAPTER 5

Breaking Integrated Circuit Device Security through Test Mode Silicon Reverse Engineering

Integrated circuits serve a wide range of purposes ranging from simple tasks in everyday products to high performance or security critical applications. Despite their abundant occurrences, the involved manufacturing processes are still considered challenging. Increasing design sizes and a constantly decreasing feature size have raised the probability of producing defective or faulty dies. To avoid additional costs for further processing defective dies, it is important that IC test modes such as scan tests [90] are inserted into the design (*Design for Testability - DFT*). In multiple stages of the manufacturing process, the chip can thus be tested and eventually discarded in case of production defects. However, from a security perspective, these testing modes can lead to a number of threats [24] and jeopardize the overall system security. For instance, only recently a backdoor has been discovered in the high security ProASIC3 chip family that can be exploited using the JTAG Test Access Port (TAP) [100]. While at least some hardware test interfaces such as JTAG Boundary-Scan Test (BST) are standardized [47], their internal test data structures and test functionalities are strictly proprietary, as they depend on the actual design that has been implemented in the integrated circuit. Especially with JTAG in micro-controllers or FPGAs, it is common that generic functions are made available to the user (i.e. for programming purposes), while the proprietary manufacturer *extensions* are well hidden and kept secret [24, 45, 100]. State-of-the-art attacks typically utilize side-channel information such as power consumption or EM emission [100, 131, 132] to find hidden commands. Similarly, scan-chain attacks extract data from the scan based test modes and use search or brute force algorithms to extract valuable information such as key material from the stream [24]. To thwart these attacks, protection mechanisms such as scan chain scrambling [42] or encryption with hard-coded keys [69, 131] have been proposed. In this paper, we show that limited effort deep silicon analysis can be utilized for proprietary test mode reverse engineering. We demonstrate our approach on a cryptographic authentication chip

in a well known game console. Our approach leads to the full disclosure of the implemented test modes, allowing us to bypass security restrictions and reveal previously kept device secrets such as the firmware, the implemented cryptographic algorithm or encryption keys. In addition, our approach allows us to circumvent proposed test-mode protection mechanisms depending on hard-coded key material or secret scrambling mechanisms [42, 45, 69, 131], since the necessary information can be gathered through silicon reverse engineering as well. Summing up, the contributions presented in this paper are as follows:

- By means of an exemplary authentication chip part of a well known game console, we show how limited effort deep silicon analysis can be used to reverse engineer proprietary test modes, leading to a complete security breach of the chip.
- We present a previously secret cryptographic algorithm we obtained by reverse engineering the extracted firmware from the device's ROM.
- We evaluate our findings by creating a full proof-of-concept implementation of the disclosed algorithm on an FPGA development board. Exchanging the proprietary chip in the game console with our FPGA implementation shows that both the retrieved algorithm and the secret keys are correct.

5.1 State-of-the-Art and Related Work

Several approaches already recognized the possibility to reverse engineer a chip's functionality by investigating test modes. However, most of them are non-invasive in nature. In [100], for example, Sergei Skorobogatov and Christopher Woods explore undocumented JTAG features using sophisticated side channel attacks. To reverse engineer these JTAG features, they use differential power analysis and pipeline emission analysis in combination with varying the content of data fields for the instructions. They propose a mitigation technique that adds noise and better protective shielding. Since our approach uses silicon reverse engineering of the scan chain logic instead of side channel attacks, it is not affected by these countermeasures.
In [24], Jean Da Rolt et al. demonstrate an attack on single and multiple scan chain structures with or without response compaction on an AES crypto-core. This includes the AES-specific limitation that only flip-flops that belong to the round register are supposed to flip between two plaintexts. The general principle of the used attack consists in observing the data stored in the round-register after the execution of the first round for several known plaintexts by means of scan-out operations, and then, from these observations, to derive the secret key. This approach requires read access to (optionally compacted) internal registers via the scan chain which might be hidden or cryptographically secured. In contrast, our approach does not share the requirement that the only changing flip-flops would be those of the round register and also evaluates restricted access to internal registers.
In [42], David Hély et al. suggest the use of scan chain scrambling in combination with a random number generator to secure the scan chain. This approach depends on the attacker not being able to predict the random numbers for the scrambling algorithm. The security of these concepts can be described as "security-by-obscurity" approach. As such, this method can easily be reverted

by reverse engineering and/or influencing the random number generator. Such approaches are not sufficient to effectively protect a chip against reverse engineering but merely raise the bar in terms of required time and effort.
Invasive reverse engineering was demonstrated in [82] by Karsten Nohl et al. In this work, the authors discuss automated reverse engineering of Mifare Classic RFID tags. They suggest that obfuscation of the implementation could increase the complexity of circuit detection, but did not investigate this type of mitigation in detail. Although we use a similar method of reverse engineering, we specifically target the testing logic and not the implementation of the whole chip.

5.2 IC Design and Test Modes

When considering digital integrated circuits, their design process typically starts on the logic level, for instance by using a hardware description language such as VHDL or Verilog. The design then undergoes a number of behavioral and timing related functional simulations to ensure that the implementation is working according to its specification. In the next step, the design is mapped to an actual process technology and manufacturing process. This may involve using existing libraries that contain typical design elements such as memory cells, logic gates or bonding pads. Once the design is complete, the manufacturing process can start. Using a silicon substrate wafer, layers involving different materials (such as polysilicon, tungsten, aluminium or insulation oxides) are deposited and selectively removed using lithographic processes to transfer the fabrication patterns onto the surface, followed by different types of wet and dry etching techniques [125]. Each of these manufacturing processes can potentially lead to subtle defects that render individual dies on the wafer unusable. To avoid additional costs (i.e. packaging, bonding, etc.) from further processing these defective dies, it is important that prior to the IC manufacturing process, IC test modes such as scan chains are inserted into the design. ASIC designers can leverage dedicated scan chain insertion tools for this task. Common test modes are scan chains, JTAG test functions, Built-In Self-Test (BIST) or proprietary test modes, for instance [90]. After fabrication during the wafer test, this allows the manufacturer to use probe cards in order to connect to bonding or test pads of the individual dies on the wafer. By supplying test vectors and observing the responses, it can be determined if the chip is working properly. Since chip packaging and bonding might cause failures on their own, manufacturers tend to make these test pads available on the outside of the IC package as well. In the IC datasheets, these pins are then often labeled as “do not connect”, “test” or “reserved”. If the test mode is enabled, testing functions such as scan tests [90] can be performed, while otherwise, the device functions in its normal intended behavior. Focusing on the security implications, the following sections provide an overview of typical test modes for digital integrated circuits.

Scan Chain

Any synchronous digital logic can be described on the Register Transfer Level (RTL). Based on the input data of a certain register (i.e. a set of flip-flops), the logic circuitry produces a well-defined output and stores it in subsequent registers. The less combinational logic is between these registers, the faster the overall system can be clocked. In order to test the overall system for

manufacturing defects, a common technique is called *scan chain insertion* [18]. The basic idea is to exchange the flip-flops in registers between combinational logic with *scan flip-flops*. These flip-flops are connected to each other in a chain to form a single large shift register.

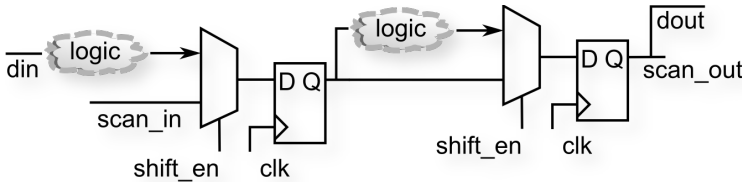

Figure 5.1: Simplified Scan Chain

Figure 5.1 shows a simplified scan chain design. During normal operation, data is supplied at `din` and processed through the combinational logic. The result goes through the multiplexer and is stored in the subsequent flip-flop. In the following clock cycle (`clk`), the data can be processed by the next logic block and eventually the computation result will be available at `dout`. When the scan chain testing mode is enabled (`shift_en`), the flip-flops form a large shift register instead. Test data can be supplied on `scan_in` and shifted into the scan chain. If the test mode is temporarily disabled and the system is clocked, it will process the test data. In the next step, the test mode can be re-enabled and the internal computation results can be shifted out of the scan chain (`scan_out`). Comparing the expected test data with the actual test results allows the manufacturer to verify the system and to precisely identify possible manufacturing defects. On the downside, the approach also allows attackers to insert arbitrary data into or extract sensitive information from the design. In [24] for instance, Da Rolt et al. demonstrate that even with counter measures such as response compaction in place, scan chains can be abused to extract sensitive encryption key material. However, without knowledge of the internal scan chain design structure, attackers can merely guess the purpose and meaning of the scan chain data they observe. Especially with protection mechanisms such as scan pattern watermarking, spy flip-flops, output obfuscation [24] or scan flip-flop randomization [42] in place, practical attacks based on statistical or brute-force based approaches get significantly harder.

JTAG Test Functions

JTAG (Joint Test Action Group) is an often used synonym for the IEEE Standard for Test Access Port and Boundary-Scan Architecture [5]. The basic idea is to have a common Test Access Port (TAP) accessible from the outside of the device. Typically, the TAP interface comprises the following signals: Test Data Input (`TDI`), Test Data Output (`TDO`), Test Clock (`TCK`) and Test Mode Select (`TMS`). Similar to the previously explained scan chains, data can be shifted in and out of the device using the clock `TCK` as well as the `TDI` and `TDO` pins, respectively. By connecting the `TDO` output of one device to the `TDI` input of the next device, JTAG also allows multiple devices to be chained together. At the heart of JTAG within the TAP-controller, there is a state machine (Figure 5.2) that can be controlled with the `TMS` pin. In general, a user can supply commands to the Instruction Register (IR) and exchange data with the test mode using the Data Register (DR). While the Test Access Port (TAP) and some essential commands (such as the device identification using the *IDCODE* command) are standardized, manufacturers typically implement their own JTAG instruction extensions. Naturally, these proprietary extensions are kept secret and can again lead to the full disclosure of internal device states. For instance, Skorobogatov et al. recently used advanced Side Channel Attacks (SCAs) to find hidden JTAG commands that led to the disclosure of a potential backdoor inside a high-security chip [100]. On the other hand, finding undocumented JTAG instructions might not be sufficient, as a single manufacturer testing command might comprise multiple JTAG instructions and data transfers. While using side channel information to uncover secret JTAG testing commands is promising, it is still likely that possible commands and command interdependencies are missed during analysis, for instance due to a high amount of noise in the measurements.

Built-In Self-Test (BIST)

Built-In-Self-Test (BIST) is another common test mode in modern IC designs [78]. The basic idea is to have a Test Pattern Generator (TPG), typically producing pseudorandom test data from a static seed value. The computed results based on the test data are then compared with a look-up table of known good values. If the produced patterns do not match, the device knows that an error has occured. Similar to the scan chain architecture, the results of the tests can be stored in shift registers and multiple BIST tests at different locations in the device can be combined by chaining their shift registers together. While these self tests can be implemented with proprietary interfaces, manufacturers often implement the tests as undocumented JTAG extensions [5]. Depending on the actual implementation of the self-test, the test results can leak sensitive information such as key material as well. However, attackers can only use this information if they can determine how the BIST can be enabled, how it works and what kind of data it includes.

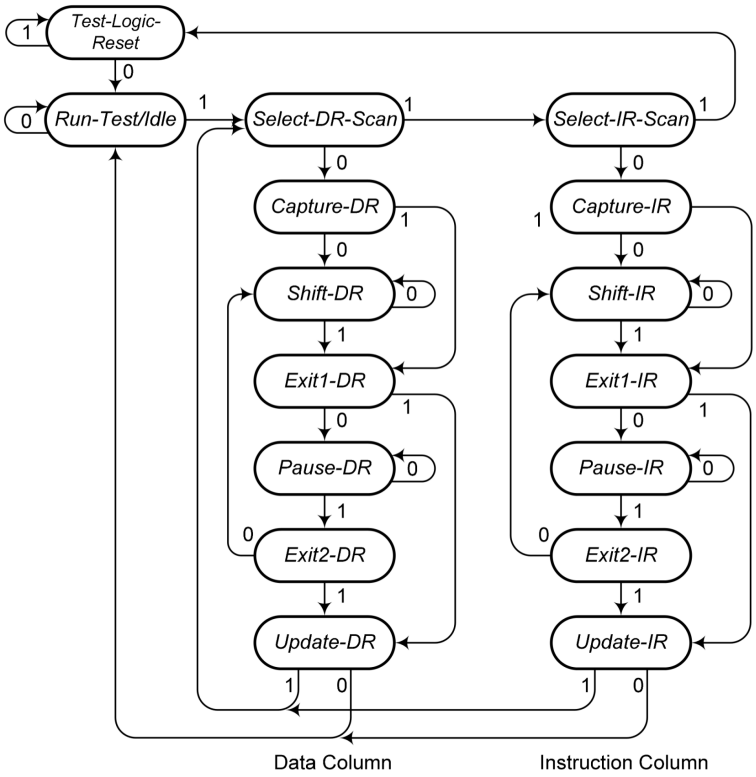

Figure 5.2: JTAG State Machine [5]

Proprietary Test Modes

Although other test modes such as JTAG or scan chains are a de-facto standard supported in a wide range of IC design tools, manufacturers can still choose to implement their own proprietary test modes. Possible reasons to choose this option are smaller sizes, less design overhead, reduced costs or certain requirements that can not easily be met by prevalent testing modes. Compared to standardized test modes like JTAG or scan chains, proprietary test modes are much harder to reverse engineer with non-invasive methods. Moreover, for more complicated and complex test modes, non-invasive techniques like sniffing, signal injection or side-channel attacks are even less promising. In these cases, deep silicon analysis is clearly the better choice.

5.3 Reverse Engineering of IC Test Modes: A Case Study of a Game Authentication Chip

In this section, we show how to identify and reverse engineer a proprietary test mode. To this end, we use a stock Nintendo 64 game authentication chip (referred to as CIC - Checking Integrated Circuit) and describe how it is possible to obtain device secrets like the software implementation and the key material stored in the device's ROM. The same methodology can of course be used to investigate all of the previously introduced test modes.

Hardware System Analysis and Pin Identification

As a first step of every reverse engineering attempt, it is important to determine the pin-out and the functionality of the chip within the system. At the example of the Nintendo 64 system, we opened up a game cartridge as shown in Figure 5.3.

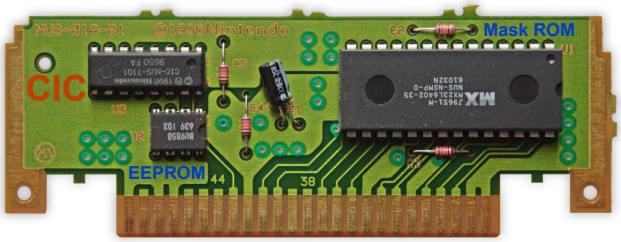

Figure 5.3: Nintendo 64 Game Cartridge PCB

On the left upper side of the picture, the CIC chip is visible. Since the chip is strictly proprietary, no official information such as a chip data sheet or information about the pinout is available. Judging from the printed labels, it can be determined that the chip on the lower left side is a serial EEPROM with 4kbit memory capacity originally manufactured by Rohm while the large chip on the right is a 4Mx16 mask ROM memory manufactured by Macronix holding the game code. Tracing the circuit lines allows us to identify the potential IC supply pins, the pins used for communication between the game console and the cartridge as well as unused pins (either floating or tied to the power rails) which are good candidates for testing mode pins. Without further analysis (such as measuring the current consumption), unused logic pins tied to the Vdd or GND power rails can not be distinguished from power supply pins. Since we intended to decapsulate the chip, we did not perform such measurements.

To gain more insight into the behavior of the chip during run time, we used a logic analyzer to capture the signals (Figure 5.4). The results show that the pin attached to D_0 is apparently the reset pin, while D_2 is the clock signal coming from the console. D_1 is one of the unconnected pins of the CIC chip and outputs a clock signal at half the frequency of the clock signal supplied from the console (D_2). Its correlation with data transfers revealed D_3 as a strobe signal for serial data communication, while the actual data is on D_4.

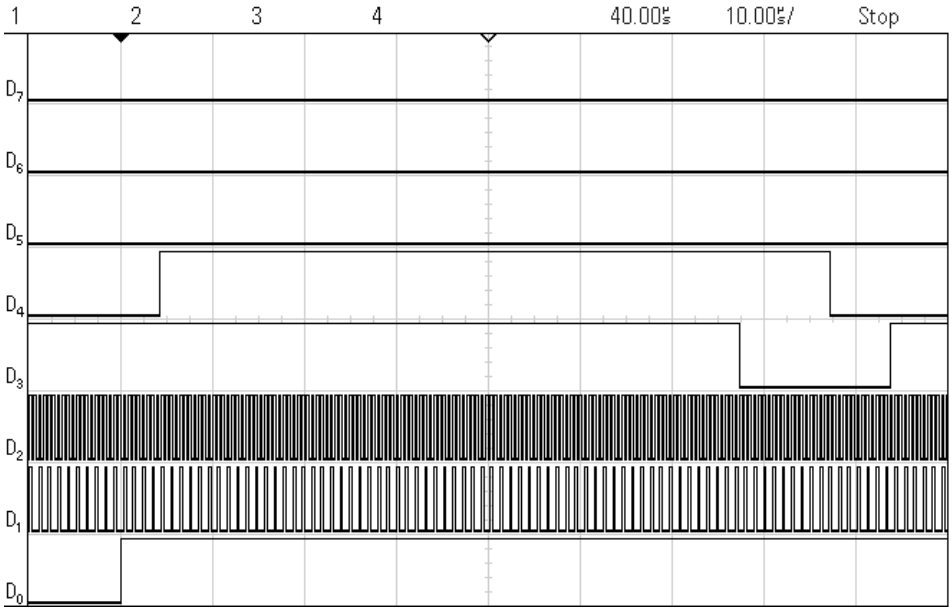

Figure 5.4: Logic Capture on the CIC Chip Signals During Initialization

IC Decapsulation

In the next step, we desoldered 5 CIC chips from game cartridge PCBs and prepared them for the decapsulation process by mechanically removing as much material from the package as possible. Since the die is usually in the middle of the IC package, we used a Dremel tool to mill a flat cavity into the top of the DIP (Dual In-line Package) packages and cut off the left- and rightmost parts of the packages as well. Moreover, we removed all pins from the packages.

Prepared this way, we performed two etching rounds in concentrated sulphuric acid at a temperature of $170\,^{\circ}\mathrm{C}$. Although decapsulation methods for epoxy IC packages typically involve the use of fuming nitric acid (e.g. WFNA - white fuming nitric acid), we opted for concentrated sulphuric acid as it can be obtained easily and at low cost even by hobbyists. Our decapsulation efforts thus focus on practical low-cost approaches that can be done with equipment available for less than 100 US$. The purpose of the first etching round is to fully remove the epoxy package. With the packages we had, the acid started to visibly attack the epoxy at roughly $140\,^{\circ}\mathrm{C}$. Since the acid in the beaker quickly turns black because of the dissolved epoxy, it is hard to visually judge how much of the package has been dissolved already. After an etch time of 10 minutes, we removed the die from the beaker and rinsed it in demineralized water. At that point, the die was already fully exposed, but still contained partial epoxy residues hindering optical analysis. The purpose of the second etch round (i.e. the *clean etch*) is to remove the epoxy remainders without causing new ones from the already dissolved epoxy in the acid bath. After an etch time of 5 minutes, we removed the die from the beaker and cleaned it in an acetone bath in an ultrasonic cleaner. The result was a clean die with some of the bonding wires still attached. Using ultra-fine tweezers and a razor blade, we removed them as well.

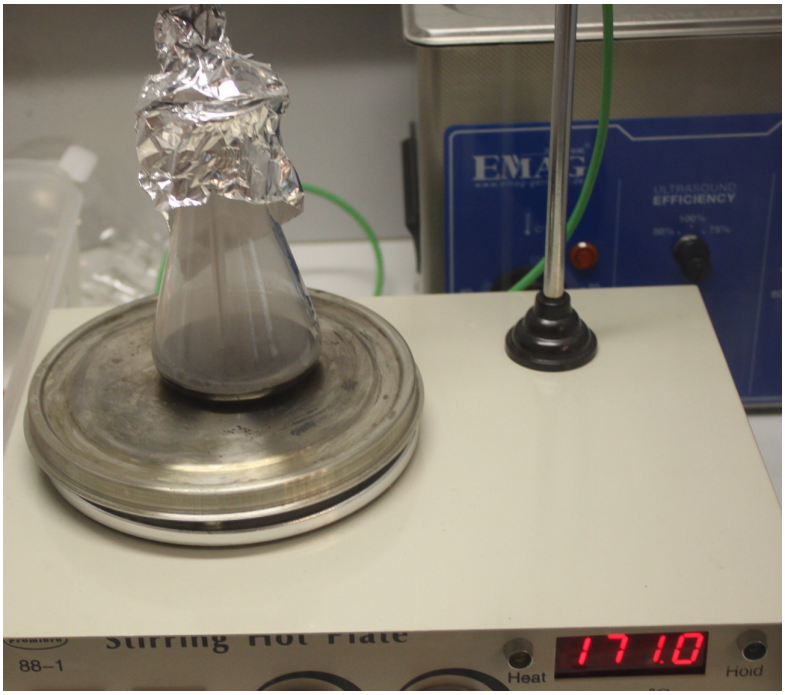

Figure 5.5: Decapsulation in Concentrated H_2SO_4

Imaging

In a reflected light microscope with motorized stage and digital camera, we took 19x27 (i.e. 513) tiled pictures and stitched them together using a custom script and the well-known Hugin software to obtain a detailed 87 Megapixel image. Although a microscope with motorized stage is highly convenient to take tiled pictures, the motorized stage is not a requirement. Usable reflected light microscopes without motorized stage are now well in the price range for many hobbyists (i.e. for less than 1,000 US$ on eBay). In our image analysis, we carefully traced the signals from the bonding pads to associated logic blocks within the die. This also allowed us to determine whether the pins that were previously tied to the power rails are supply or potential test-mode pins. In addition to the optical microscope, we used our lab's Scanning Electron Microscope (SEM) whenever greater detail pictures were helpful. For SEM application, we did not prepare the dies in any special way other than cleaning. Similar to reflected light microscopes and with second hand price tags below 5,000 US$, SEMs are now becoming affordable for hobbyists or hackerspaces as well.

A heavily scaled down and commented version of the CIC chip is shown in Figure 5.6. The numbers on the side represent the pin numbers of the CIC DIP Package. The image allowed us to spot typical logic design blocks such as ROM memories (marked red with letters *B* and *E*), SRAM (marked black with letter *C*) as well as a CPU (marked green with the letter *D*). Combining the information we obtained earlier with regard to potential test mode pins, we could now exclude the power supply pins as well as the clock output pin from the list of test-mode candidates. This

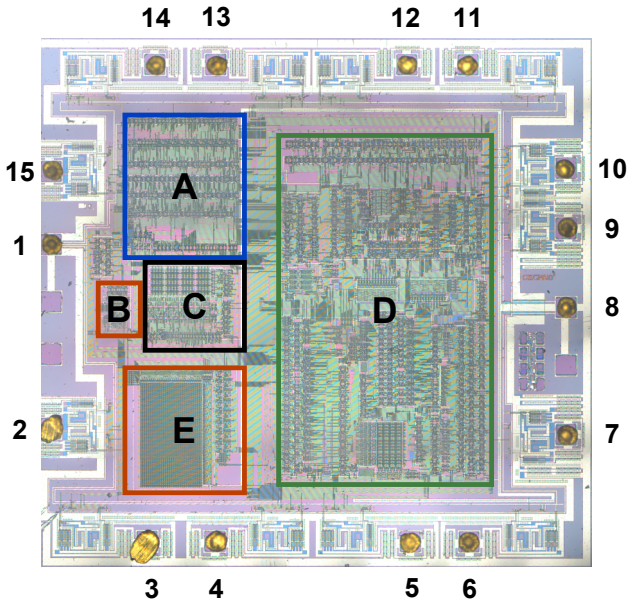

Figure 5.6: Commented CIC Die with Manufacturer Chip Label CECRN8

left us with the potential test-mode pin candidates 2 to 7. However, while the signals from pins 2 to 5 all led to a register, we could trace pins 6 and 7 to corresponding logic blocks (marked in blue with letter *A*). Those pins are thus likely to be test-mode control pins.

Detailed Test Mode Reverse Engineering

Within logic block *A*, we traced the test mode signals from pins 6 and 7 to the logic block shown in Figure 5.7.

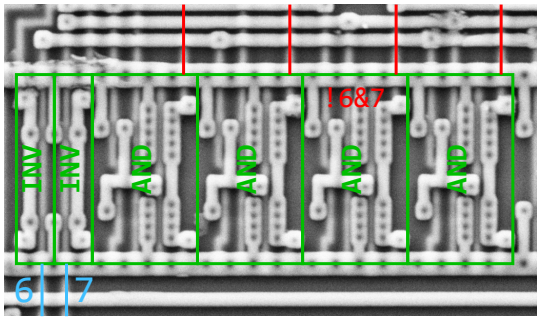

Figure 5.7: Proprietary Test Mode Logic

Both signals first go through an inverter. After that, both the inverted as well as the non-inverted signals are available and used by four AND gates to obtain all possible bit combinations from the two testing pins. Since both pins were tied to ground on the game cartridge PCB, this leaves 3

possible testing modes. Tracing these outputs further, we could see that they control multiplexers as shown in Fig. 5.8.

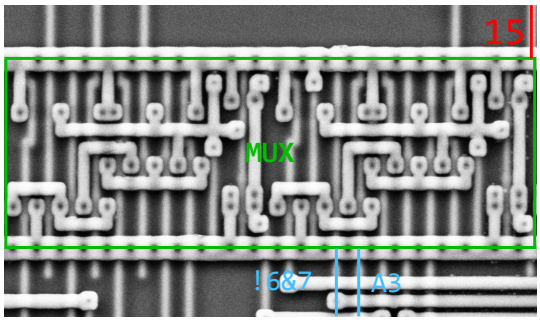

Figure 5.8: Test Mode Multiplexer

However, the source data entering these 4 multiplexers always comes from one of 3 different registers spread across the chip and, even more interestingly, the output of each multiplexer is connected to the output pins, that are otherwise used to communicate with the console. The registers are relatively easy to spot, as they are made up of equally looking CMOS flip-flops right next to each other. Figure 5.9 shows a typical CMOS flip-flop, comprising two latches and control logic.

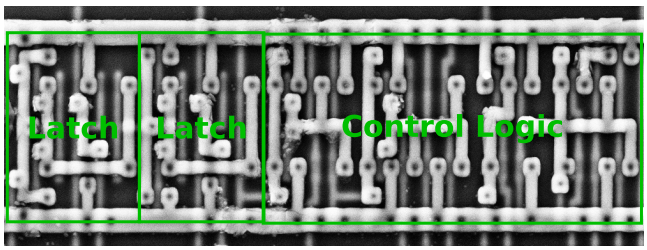

Figure 5.9: CMOS Flip-Flop comprising two Latches and Control Logic

Since not all possible test mode combinations were covered by the analyzed multiplexers, we traced the test mode signals further across the die and found similar control logic that controls the input register containing the data from the previously identified pins `2` to `5`. However, the identified logic was right next to the ROM (Fig. 5.10). Further analysis showed that the control logic allows the instruction code to be either fetched from the ROM or from the input register. In other words, selecting a testing mode always allows *arbitrary code execution*.

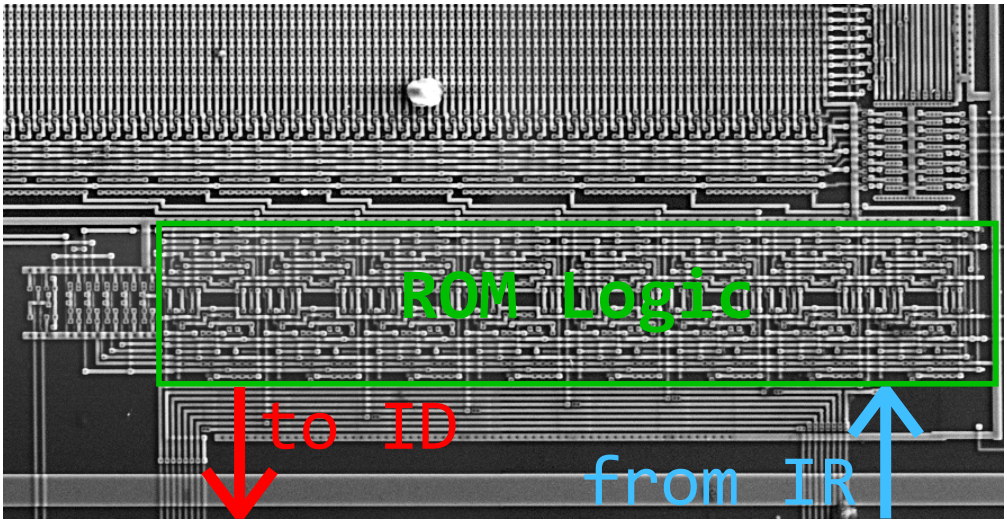

Figure 5.10: Implanted 1024x8 Bit Mask ROM with Control Logic

De-Layering

In some cases the chip's top metal layer obstructed the view to significant parts in the lower layers. For this case, we used a polishing machine with a 0.1μ polishing disc (Fig. 5.11) to remove the die's insulating layer and the metal layer below.

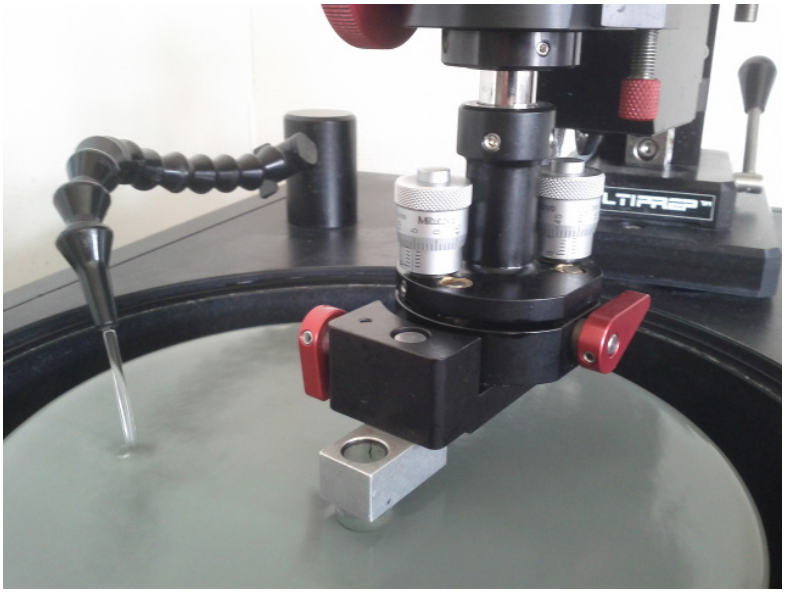

Figure 5.11: Polishing Machine with 0.1μ Polishing Disc and Water as Lubricant

One of the main challenges employing polishing techniques is to get a planarized result. Otherwise, it can easily happen that on one side of the die all layers were already removed while on

the other side the insulation layer is still intact. Although there is much room for improvement, in our approach we used an Aluminium holding piece with a glass rod attached to it that can be clamped onto the polishing machine (see Fig. 5.11). We polished the tip of the glass rod until it was even and then used heated Allied wax (i.e. a wax that can be dissolved in Acetone) to glue the die onto the glass rod with the help of a toothpick. Under an optical microscope, we used the lens focus to check whether the die surface is parallel to the surface of the glass rod. If not, we heated up the holding piece and repeated the adjustment step. Using a total of 2 decapsulated dies for de-layering, we finally obtained usable results. In the subsequent process of test mode reverse engineering, we resorted to manual image analysis as we only had a strongly limited number of dies and our polishing results would have been problematic for automated pattern recognition approaches. Figure 5.12 shows a part of the CPU's Instruction Decoder (ID) unit with the insulation and top metal layer removed. The signals coming from the left of the picture originate from the ROM Logic (Fig. 5.10) while each of the signals leaving the ID unit at the bottom represents one CPU instruction. The partial reverse engineering of the CPU's ID unit allowed us to verify some publicly available bits of information. The CIC chip, for instance, is supposed to use the Sharp SM5 4-bit CPU core. By comparing the instruction opcodes used in the ID unit with the opcodes found in the SM5 data sheet, we found this information to be accurate. Due to die markings and the fact that manufacturers tend to re-use existing CPU cores (such as the Sharp SM5 core in this case), we believe that architectural information for proprietary ICs is often available to attackers. If this information is not available, the CPU architecture could still be determined through reverse engineering approaches similar to our initial approach that targeted the CPU's instruction decoder (ID) unit. The necessary effort would be considerable though.

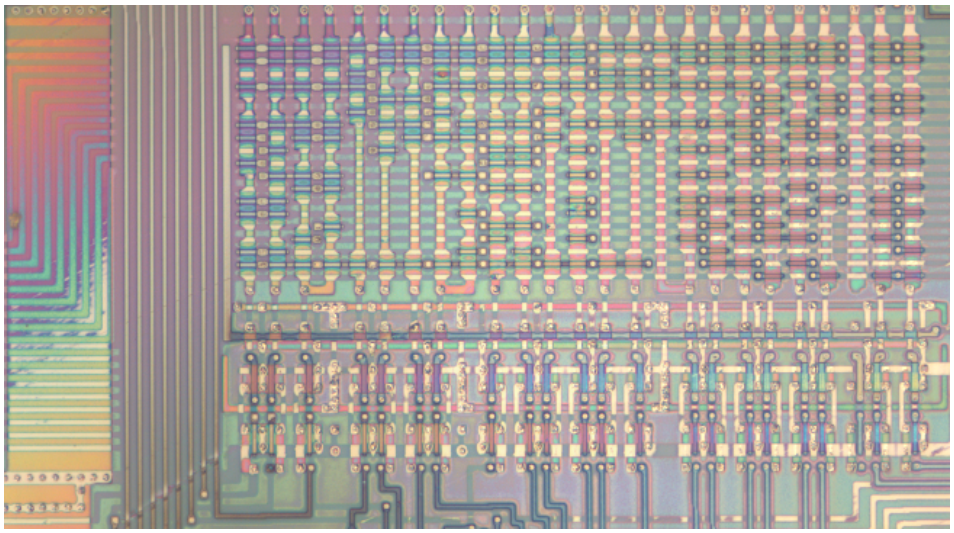

Figure 5.12: Poly Layers below M1 Metal Layer of CPU Instruction Decoder Unit

ROM Firmware Extraction

Combining the information we obtained on the proprietary test modes in the CIC chip so far, allowed us to produce the complete chip pinout (Fig. 5.13) as well as a deeper knowledge of how the test modes can be utilized.
The test mode allows us to either output the content of the SM5 AREG accumulator or the PC program counter register. If enabled, the corresponding output can be made available on SM5 Port 2 (i.e. on pins 12 to 15). Similarly, the enabled test mode allows arbitrary code execution by supplying CPU opcodes on SM5 Port 5 (i.e. on pins 2 to 5). In that case, the CPU fetches its instruction from the input register of Port 5 instead of the ROM (Fig. 5.10). Since an instruction has a size of 8 bit while the input register is just 4 bit wide, it takes 2 clock cycles to load a full instruction. Utilizing these powerful discovered test modes, we could finally start to extract the content of the whole ROM firmware and reverse engineer the hitherto secret authentication algorithm and key material. During normal chip operation, we could observe the system clock on the TIO pin. We hooked the CIC chip to an FPGA board and performed several tests that finally led to a full ROM dump. First, we enabled the test mode to test arbitrary code execution. We observed the continuous clock output on the TIO pin and supplied the SM5 HALT instruction (0x77) on Port 5. Subsequently, the CPU executed the HALT instruction and the observed clock output on the TIO pin was halted. Similarly, we used the STOP instruction (0x76) to determine the correct ordering of the input nibbles. With the correct ordering, the instruction executed as intended and halted the clock output on the TIO pin just the same.
In the next step, we wanted to observe the output of the AREG accumulator register on Port 2. Utilizing the LDX (load constant into accumulator) instruction, we loaded a constant into the AREG accumulator register. However, at that point we could not see the output on Port 2 as the port is configured as input port after reset. By setting the corresponding bits in the SM5 directional register, we configured Port 2 as output port and, subsequently, the test mode worked as expected.
To extract the full content of the ROM, we exploited another implementation detail of the test mode. A JUMP instruction comprises two bytes. The first nibble is the instruction while the subsequent bits represent the 12-bit destination address of the jump. By toggling the test mode accordingly, this allowed us to supply the first byte (i.e. the JUMP instruction followed by a zero nibble) to the CPU, while the second byte was loaded from the ROM.
Executing the instruction thus led to a jump to an address that corresponds to the *opcode byte* of the according instruction in the ROM. Using the test mode, we can also observe the PC program counter register and thus determine what this jump target was, and thus, what the ROM contained at this position. Furthermore, by executing ordinary manual jumps via the test mode, we could target arbitrary regions in the ROM. In other words, we used the internal program counter to read the complete ROM content in a byte-by-byte fashion. Ultimately, this technique allowed us to create a simple implementation on our FPGA board that dumps the full ROM content (including the cryptographic algorithm and the key material) and outputs it to the UART port. An attached PC with a serial port was used to finally write the ROM dump to a file for later analysis.
In contrast to our approach, we would like to note that there are other ROM extraction techniques such as optical readout as well. However, the usable extraction techniques strongly depend on the ROM implementation type. For instance, so called *mask-ROMs* have their memory configuration

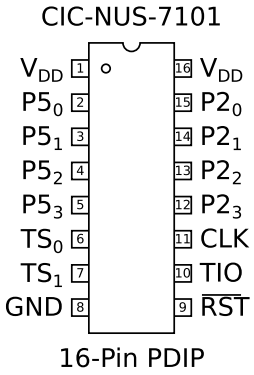

Figure 5.13: Pin-Out of the CIC Chip

in the physical layout of the ROM and can thus be read out optically. In contrast, other ICs (like the CIC chip in our case) utilize ROM implementations such as ion or laser implanted ROMs which are programmed after fabrication through careful alteration of the transistor threshold voltages (for instance, by means of an ion or laser beam). As in implanted ROM memory cells the 0 and 1 bits can not be optically distinguished from each other, optical readout is not feasible without further deprocessing and analysis steps (such as performing a so called *dash etch* process that changes the color of the active chip areas based on their doping).

5.4 Evaluation and Discussion

In this section, we show how we evaluated our approach by reconstructing the previously secret CIC algorithm from the obtained ROM firmware, creating our own implementation of the algorithm on an FPGA board and testing it with the unmodified gaming console.

CIC Algorithm

Taking the full ROM dump we obtained in Section 5.3, we manually disassembled the firmware and extracted the CIC game authentication algorithm shown in Listing 5.1.
On startup, the CIC IC initializes the RAM. The first 2 nibbles for RAM initialization are supplied from the console while the rest is a static key[1].
It then reaches an infinite loop, mutating the RAM and sending hashes of it to the console. The console verifies the incoming endless bit stream and freezes if the stream is different from the internally generated one.

[1] As we do not endorse nor support software piracy, we're only disclosing the algorithm, but not the static key material.

```
1  void cic_round(uint4_t m[16])
2  {
3     uint4_t a, b, x;
4
5     x = m[15];
6     do
7     {
8        m[1] += x + 1;
9
10       b = 6;
11       if (15 - m[3] > m[2])
12          b += 1;
13
14       m[b - 3] += m[3];
15       m[3] += m[2] + 1;
16       m[2] = ~(m[2] + m[1] + 1);
17
18       a = m[b - 1];
19
20       if (m[b - 2] > 7)
21          m[b - 1] = 0;
22       m[b - 1] += m[b - 2] + 8;
23       m[b - 2] += m[b - 3];
24
25       do
26       {
27          a += m[b] + 1;
28          m[b] = a;
29          b += 1;
30       } while (b != 0);
31
32       x -= 1;
33    } while (x != 15);
34 }
```

Listing 5.1: Secret Reverse Engineered CIC Algorithm

The `cic_round` function presented in Listing 5.1 is responsible for mutating the RAM `m` (16 * 4 bit). The CIC implementation calls the `cic_round` 3 times in a row on the same RAM block `m` to garble the bits. After that, it uses the 7th nibble to determine how many bits (`N`) should be sent to the console. Starting at nibble 0, it transfers the Least Significant Bit (LSB) to the console, followed by the LSB of nibble 1, the LSB of nibble 2 and so forth until all `N` bits have been sent. The `cic_round` uses a loop in line 6 to 33 to run for a given number of rounds (1 - 16) depending on the current state. In each round, most elements in RAM are modified by adding them to each other in the presented way.

FPGA Implementation

We evaluated the reverse engineered CIC algorithm by creating a full FPGA implementation on a Digilent Nexys 4 Artix-7 FPGA board. On the Artix-7 XC7A100T FPGA, our implementation takes 300 Slice LUTs and 155 Slice registers. We do not utilize any BlockRAM or DSP resources. Our testing setup is visible in Figure 5.14. On the left of the picture, we used a PC with the Xilinx Vivado FPGA design tools to program the FPGA board. In the middle of the picture, the FPGA board and the opened up game console with a game cartridge is visible. We used one of the game

cartridge PCBs where we had previously removed the CIC chip and connected the FPGA board containing our CIC implementation and the secret key instead. On the right side of the picture, we used a logic analyzer to analyze the communication between our CIC implementation on the FPGA board and the game console. As visible on the TV screen, inserting the game cartridge PCBs into the console correctly started and executed the game contained in the cartridge's ROM chip. Stopping the execution of the algorithm within our FPGA board immediately froze the game. To test whether our implementation still worked after multiple hours of game play, we invited some friends and enjoyed Mario 64 all together.

Figure 5.14: Evaluation Test Setup - Nexys 4 FPGA Board and Gaming Console

Security Tradeoff

Mitigation techniques against test-mode attacks conservatively follow a security-by-obscurity approach. Commonly proposed methods found in the literature include scan chain scrambling [42], encryption with hard-coded keys [69, 131], scan pattern watermarking, spy flip-flops, output obfuscation [24] or scan flip-flop randomization [42]. While we acknowledge their effectiveness to raise the bar for non-invasive attack scenarios, we show that they offer next to no protection against deep silicon analysis. On the other hand, there are more secure ways to protect a chip, like cryptographically signing manufacturer test mode commands and only execute them within the chip for instance. Other methods rely more on Built-In Self-Test (BIST) modes that do not leak sensitive information. These defense strategies, however, have a common drawback. Since they always cause additional costs both in design and production, they ultimately create the need to trade security/functionality for cost. Simple obfuscation techniques are easy to implement and can be used with powerful testing modes such as scan chains, but do not provide high security. On the other hand, Built-In Self-Test (BIST) modes might not provide the necessary testing

granularity while secure testing modes employing cryptographic signature checks also need potentially large cryptographic cores on the dies that increase the production costs. In fact, the logic required for signature checking (i.e. RSA, DSA, ECDSA, etc.) can be huge (i.e. cost intensive) and, depending on the chip design, it can be even bigger than the rest of the chip design on its own. How to choose in this typical tradeoff situation depends on the application domain. A hacked gaming console might be more tolerable than a reverse-engineered chip for wireless payment. Still, we recommend to include secure protection mechanisms whenever the projected costs permit. Less sophisticated countermeasures against non-invasive reverse engineering might provide a certain level of security but are completely ineffective against a motivated effort with deep silicon analysis.

5.5 Conclusion and Future Work

In this paper, we demonstrated that limited effort silicon analysis can be effectively used to reverse engineer secret test modes and break device security. Our example application of the developed techniques revealed previously secret content of a cryptographic game authentication chip. Specifically, the discovered testing mode allowed us to execute arbitrary code on the chip and subsequently dump the secret firmware and key material. While the authentication chip in a game console is not a highly critical or especially security-sensitive application, we believe that our example effectively illustrates how undocumented and proprietary testing modes can easily be discovered through silicon reverse engineering. Furthermore, we prove that most widely proposed obfuscation-based countermeasures can be circumvented without modifying the analysis approach.
As our technological reverse engineering procedure proved feasible, we plan to extend our efforts with regard to test mode silicon reverse engineering for analyzing security critical applications. The major challenge will be to overcome more sophisticated anti-reverse engineering techniques that specifically aim to protect against deep silicon analysis.

5.6 Acknowledgements

This work has been partly funded by the $(SG)^2$ project under national FFG grant number 836276 through the KIRAS security research program run by FFG and BMWFW. In addition, we would like to thank USTEM [115] at Vienna University of Technology and Trustworks KG for providing valuable tips and letting us their lab equipment. Without their support, this work would not have been possible.

CHAPTER 6

PROSPECT - Peripheral Proxying Supported Embedded Code Testing

Embedded systems are omnipresent in today's world. From small digital clocks over home appliances such as washing machines and multimedia devices to medical appliances or smart phones, embedded technology provides tremendous advantages compared to general purpose systems. One key aspect is the possibility to create tailored hardware devices to fulfill a very specific task. With exactly the right amount of memory, processing power and interfaces, embedded devices are cheaper, smaller and faster than their general-purpose computing counterparts. However, a good amount of embedded devices are aimed at functionality rather than security. In fact, recent publications have shown that the security of embedded devices is especially bad [58, 65, 88, 119]. One reason is, that security audits on embedded devices are considered to be far more challenging and time consuming than on general purpose PC systems. Considering the common case in which the security analyst has no access to the source code of the system under test, there is a broad gap between state-of-the-art security analysis techniques for PCs and for embedded systems. Liu et al. [70] and Austin et al. [6] give an overview of the wide area of vulnerability discovery techniques that are available for PC systems. The techniques range from sophisticated static analysis techniques over dynamic analysis and fuzz testing to advanced dynamic taint analysis and symbolic or concolic execution [17, 94]. However, for embedded systems, the situation is different. Mainly due to custom proprietary hardware, undocumented peripherals and strict system limitations, the prevalent vulnerability discovery techniques are still based on static analysis [13, 59, 118].

At the same time, using the wide range of dynamic analysis or taint analysis and symbolic execution tools is in general not possible due to the limitations of the embedded system under test. One solution would be to take the investigated application from its original context and run it in a virtual machine that provides the necessary resources and facilities for a full dynamic evaluation. To emulate the embedded device, however, the connected peripheral hardware needs to be available from within the virtual machine as well. The usual way is to emulate the peripheral hardware. Yet, for proprietary hardware this is not possible due to the following reasons. First,

the analyst would need comprehensive information on how all peripheral hardware devices work in order to emulate the hardware behavior in software. Since peripheral hardware is likely to be proprietary, this information is not available and, subsequently, the analyst can not emulate the hardware. Second, even if the information is available to the analyst, adding full support for new peripheral hardware components to a virtual machine implementation is not an easy task. It is likely that the implementation would take the analyst a tremendous amount of time that renders the whole dynamic security analysis infeasible.
To amend this problem, we take a different approach and introduce PROSPECT, a proxy capable of tunneling arbitrary peripheral hardware accesses from within a virtual machine to the embedded system under test. The result is a virtualized execution environment for embedded software implementations with a completely transparent connection to the actual peripheral hardware components of the system under test. PROSPECT thus enables the analyst to leverage any powerful dynamic analysis techniques of her choice to discover vulnerabilities on embedded devices with minimal effort. We developed and continuously improved PROSPECT over a duration of more than 10 months during which our system evolved. In addition, we conducted a case study to prove the effectiveness of PROSPECT and used the system to undertake a full scale security analysis of a widely used proprietary fire alarm system in the building automation domain. Summing up, the contributions presented in this paper are as follows:

- We introduce PROSPECT, a transparent proxy for tunneling peripheral hardware accesses from within a virtual analysis environment to the embedded system under test. Our system can overcome prevalent analysis limitations by enabling dynamic instrumentation inside arbitrary analysis environments.
- We provide a MIPS based proof-of-concept implementation that has continuously evolved over a duration of more than 10 months.
- We evaluate and discuss our approach with a detailed analysis of the system's performance and usability.
- We utilized PROSPECT to conduct a case study by running a full-scale security audit of a widely used commercial fire alarm system in the building automation domain showing that PROSPECT is both practical and usable for real-world application.

6.1 State-of-the-Art and Related Work

When dealing with security analysis on embedded systems, most research approaches use static analysis to achieve their goal. For instance, Khare et al. presented some of the key problems that need to be faced when using static analysis techniques on a large embedded code base [59]. In their work they focus on the static analysis of source code to improve the overall security of embedded systems.
In contrast, Ramakrishnan and Gopal do not require access to source code as their static program analysis techniques run on embedded binaries [118]. However, they do not focus on embedded security or vulnerability discovery.

In [96], Zili Shao et al. introduce a mixed hardware/software system to check for and protect embedded systems from buffer overflow attacks. Their system works during program execution, but is more focused on vulnerability protection than on vulnerability discovery.
In [108], Sumpf and Brakensiek introduced device driver isolation within virtualized embedded platforms. The approach presented here can be considered the most closely related system compared to PROSPECT. The authors created device drivers with a generalized interface to provide homogeneous access for virtual machines. In contrast to PROSPECT, however, the implementation of the driver must be known. Furthermore their system is limited to L4 microkernels and not suitable for unknown peripheral devices.

6.2 Challenges in Embedded Security Analysis

Assuming that the reader is familiar with the general field of information security, in this chapter, we briefly outline binary code analysis and highlight fuzz testing as exemplary, widely established techniques to discover software vulnerabilities. We point out, that dynamic analysis is one of the key requirements for efficient fuzz testing as well as for manual in-depth analysis approaches usually done as soon as fuzz testing discovers a potential security vulnerability. After presenting dynamic analysis techniques for PC systems, we continue by providing a general overview of how typical medium to large scale embedded systems are made up and why the presented dynamic analysis approaches are frequently not applicable to embedded systems. Besides, the chapter shows why the approach PROSPECT takes is promising as it can overcome the described challenges and enable dynamic analysis in general, regardless of the analysis limitations on the system under test.

Binary Code Analysis

Vendors are typically profit-driven and try to push their newest software products to market as soon as possible. Depending on their efforts to avoid software vulnerabilities, a released software implementation may contain numerous security flaws such as stack smashing or use-after-free vulnerabilities [95]. At that point, an arms race between attackers and the vendor begins. Attackers try to exploit the vulnerabilities for their own ill-gotten gain such as industrial espionage, spreading malware or setting up botnets [57, 107] while vendors try to patch newly discovered bugs.
For proprietary software implementations, the source code is usually not available. Thus, in order to discover vulnerabilities in these implementations, security analysts need to rely on techniques that can be applied to binary code. In a recent survey [70], Liu et al. describe a number of common techniques to discover software vulnerabilities. While analysts can resort to static analysis that does not require the execution of the program under test, static analysis suffers from a number of drawbacks hindering penetration tests. For instance, object orientated code makes frequent use of function pointers that are hard to resolve, if the program is not being executed. With dynamic analysis, the program is being executed and the analyst can trace and instrument the current execution path of the program under test. However, unless advanced techniques such

as multipath exploration [79] are employed, the analyst needs to generate different program inputs to analyze different execution paths.

Fuzz Testing: A Common Technique to discover Software Vulnerabilities

Generating different program inputs to reach different execution paths is also one of the key ideas of *fuzz testing*, a widely established technique to discover software vulnerabilities [7, 27, 70]. With fuzz testing, input data to the program are generated automatically either at random or by mutating previously obtained program input. At the same time, the analyst can employ dynamic analysis to monitor the program execution and detect program anomalies such as crashes, illicit memory accesses or endless loops causing high CPU utilization. If an anomaly is detected, the generated input data are likely to have caused the abnormal behavior. This is a starting point for a more thorough manual program analysis, usually also within a dynamic analysis environment. Practical results [7, 27, 70, 95] have shown that fuzzing is both a viable and established technique to discover software vulnerabilities. However, since fuzzers can be highly application specific, it might be necessary to implement new fuzzing tools for each penetration test. Also, we would like to stress that although fuzz testing is widely used, it is not the only technique to discovery software vulnerabilities efficiently. One key observation at this point is that dynamic instrumentation is required for both efficient fuzz testing and the manual analysis that is usually done after the fuzzer discovers a potential security vulnerability.

Dynamic Code Analysis on PC Systems

In general, a PC system can be divided into hardware, an operating system (including kernel and drivers) and software applications. The analyst has the freedom to dynamically instrument any of these layers. The easiest way to instrument a program is to debug it with a state-of-the-art-debugger such as *gdb* or *Ida Pro*. The drawback here is that the program can easily detect that it is being instrumented and behave differently. For instance, the program might just exit instead of performing its usual functionality. On the next level, the analyst can instrument the operating system to analyze the program's behavior. For instance, CWSandbox [127] uses this approach by hooking the operating system libraries. This allows the analyst to trace the behavior of the application, but at the same time hinders typical techniques for debugging (e.g. single stepping through code). On the lowest level, the analyst can instrument the hardware using Virtual Machine Introspection (VMI) [38], which makes it hard for the investigated software to detect that it is being analyzed. Although the target of the analysis is the application itself, the downside of this approach is the need to analyze the surrounding operating system as well. Therefore, the necessary effort is higher than applying a regular debugging technique.

A typical Embedded System

Embedded systems can be divided into small, medium and large scale embedded systems [88, 97]. Depending on their size, their system configuration can differ tremendously. Small scale embedded systems such as electronic toys, digital clocks or pocket calculators are built around strongly resource constrained microcontrollers. Typically, there is no operating system

and the firmware of these systems comprises a single program that is contained in an on-chip Flash memory. In contrast, medium and large scale embedded systems such as smart phones, wireless routers or field level components of SCADA systems are based on more powerful controllers. Typically, they run a customized operating system (e.g. Linux) and the product-specific implementation of an embedded product often comprises custom kernel code, drivers and several applications. At the heart of these systems commonly lies a powerful System-On-Chip (SoC) controller that includes a CPU, ROM, SRAM and a number of internal peripherals and I/O controllers.

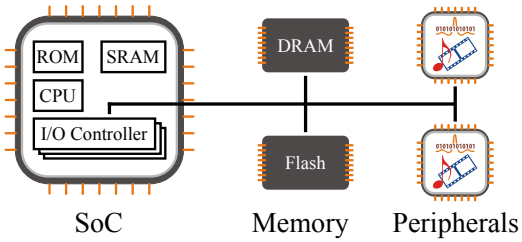

Figure 6.1: A Typical Medium to Large Scale Embedded System

A typical separation of components is shown in Figure 6.1. Upon power-up, the CPU in the SoC controller will execute the first-stage bootloader code contained in internal ROM and perform low-level initializations. After that, the SoC can access external memories (such as Flash and SDRAM) to boot into a second-stage bootloader and, consequently, into the operating system (OS) kernel. At that point, the OS can load a number of additional drivers to support external peripherals and then start the product specific processes. While the general operation of embedded systems is similar to PC systems, it is the external peripherals that make embedded systems so special. External peripherals are typically customly designed by the system manufacturers ranging from product specific sensors and actuators to custom communication interfaces. Taking modern smart phones as an example, such external peripherals could be charging controls, wireless radios, GPS receivers, magnetic or accelerations sensors, driving circuits for the vibrating alert, speech compression DSPs and many more. These external peripherals are what actually transforms an off-the-shelf SoC system into a valuable everyday product.

Challenges of Dynamic Code Analysis on Embedded Systems

In contrast to PC systems, employing dynamic analysis techniques on embedded systems can be more challenging. Typically, embedded devices are resource constrained, access to the file system is limited and the kernel's functionality and tools available on the device are just a minimal set of functions necessary for the product to operate properly [88]. The main reason for these constraints is that including additional functionality on the embedded systems would result in increased embedded resource requirements and ultimately in higher manufacturing costs. From this perspective, the presented analysis approaches for PC systems are hard to apply to their embedded counterparts:

1. Using a debugger to instrument the program is only feasible if the OS kernel includes debugging support (e.g. through `ptrace()` in the case of Linux). Running a state-of-the-art debugger on the system might not be possible due to resource constraints (e.g. in terms of memory consumption) or due to missing support (e.g on legacy systems or on systems where `ptrace()` support was not compiled into the kernel to save memory space).

2. Instrumenting the operating system would require kernel modifications or loading custom kernel modules. Embedded systems often run customized minimal kernel configurations to keep resource consumption and boot-up delays low. As a result, instrumenting the operating system might not be feasible.

3. Instrumenting the hardware would require not only virtualization of the system architecture, but also of all the necessary peripheral devices. However, as peripheral devices and their drivers are often proprietary, the information required to emulate them might not be publicly available. Besides, writing emulation code for all peripheral hardware devices would cause a tremendous overhead, considering that the analyst's goal is dynamic code analysis of only a small set of programs.

These challenges show that while on PC systems there is a wide range of established and well working vulnerability discovery techniques, the situation is different on embedded systems. In theory, all of those techniques could be applied to embedded systems as well. However, practically, embedded systems frequently lack support for these techniques and thus make it much harder to discover software vulnerabilities. We believe that this is also the reason, why static analysis techniques are still so prevalent for those systems.

6.3 Peripheral Device Forwarding

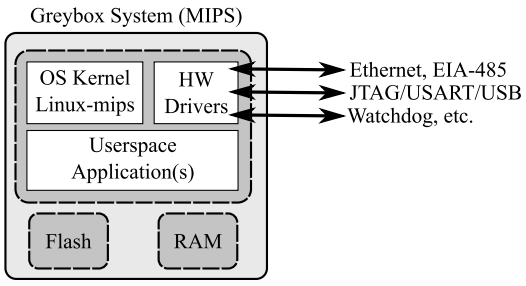

Figure 6.2: A typical Greybox System Example

Figure 6.2 shows a typical greybox embedded system example from a security analyst's point of view. The analyst's goal is to test one or more userspace applications on the embedded system for security vulnerabilities. This could, for instance, be a network daemon that is exposed to external attackers over a network connection. Yet, due to the challenges portrayed in Section 6.2, the analyst is unable to perform dynamic code analysis on the target system.

That is, the system lacks system resources, analysis tools are not available or can not be run and the userspace application, the analyst is interested in, can not be executed in a virtual environment as the peripheral hardware is missing there. However, one key observation is that userspace applications commonly communicate through character devices with potentially proprietary drivers and, consequently, with the peripheral hardware. Also, the communication interfaces to exchange data with the driver, and therefore with the kernel, are limited and can be considered standardized.
This is where PROSPECT comes into play. The basic idea of the system is to create virtual character devices inside another physical or virtual analysis environment. PROSPECT must intercept system calls used for communication with the character device from within the operating system kernel, forward them to the appropriate device on the embedded system and execute them there. Any responses need to be fed back to the analysis environment so that the intercepted system calls can return the data from the embedded remote system. Block devices, on the other hand, are generally used to access storage media which are emulated by the analysis virtual machine (i.e. qemu) anyways. To the analyst, PROSPECT constitutes a transparent forwarding solution for character device communication and thus allows her to conduct dynamic analysis techniques that were previously infeasible. Even software running on legacy systems lacking support for state-of-the-art analysis tools can be analyzed this way. As a result, PROSPECT allows to overcome typical challenges an embedded security analyst typically needs to face today.

Character Device Access

In order to forward peripheral hardware accesses, we need to know which system calls are generally used to interact with character devices. Targeting Linux systems, we gathered information on the supported `file_operations` of all included character device drivers in three different Linux kernel versions (Linux-2.4.20, Linux-2.6.38.1 and Linux-3.4.4) by analyzing the source code of all available drivers (514 files in total). We chose these specific kernel versions to get an idea which system calls are used to access character devices on legacy systems (i.e. Linux-2.4 and Linux-2.6) as well as on current kernel versions (i.e. Linux-3.4). Table 6.1 shows how many of the character device driver source code files actually define `file_operations`. It can be seen that the number of files decreases with newer kernels. We believe that this is due to increased abstraction in the Linux kernel requiring driver authors to write less supporting code.

Linux-2.4.20			Linux-2.6.38.1			Linux-3.4.4		
files	fops	fops %	files	fops	fops %	files	fops	fops %
264	77	29.17	143	62	43.36	107	54	50.47

Table 6.1: Analyzed Device Drivers on different Linux Kernel Versions

Table 6.2 shows which `file_operations` (i.e. which system calls) are used to interact with character device drivers in the different Linux kernel versions in relation to the number character device source code files. For instance, on Linux-2.4.20, there are 77 files that define `file_operations` and, out of these, 83.12% define a custom handler for the `open` system call. Some of the system calls in older kernel versions have been superseded by newer ones.

For instance, the `ioctl` call in Linux-2.4.20 has been replaced by `unlocked_ioctl` and `compat_ioctl` for performance reasons in newer kernel versions.

Syscall	Linux-2.4.20	Linux-2.6.38.1	Linux-3.4.4
aio_fsync	-	0.00	0.00
aio_read	-	1.61	1.85
aio_write	-	1.61	1.85
check_flags	-	0.00	0.00
compat_ioctl	-	6.45	7.41
fallocate	-	0.00	0.00
fasync	28.57	11.29	12.96
flock	-	0.00	0.00
flush	14.29	-	-
fsync	0.00	3.23	3.70
get_unmapped_area	2.60	1.61	1.85
ioctl	84.42	-	-
llseek	6.49	32.26	29.63
lock	0.00	0.00	0.00
mmap	18.18	12.90	14.81
open	83.12	74.19	77.78
poll	32.47	20.97	25.93
read	68.83	82.26	85.19
readdir	0.00	0.00	0.00
readv	0.00	-	-
release	77.92	62.90	66.67
sendpage	0.00	0.00	0.00
setlease	-	0.00	0.00
splice_read	-	0.00	0.00
splice_write	-	0.00	0.00
unlocked_ioctl	-	51.61	50.00
write	62.34	50.00	55.56
writev	0.00	-	-

Table 6.2: Usage of Linux `file_operations` in Character Device Drivers (Percentage)

In theory, PROSPECT could forward any of the system calls visible in Table 6.2. However, for performance reasons, it is beneficial to handle some of those calls locally. In fact, PROSPECT could execute system calls such as `flush`, `sync`, `fasync` or `aio_fsync` locally, if device access on the remote device is kept synchronized. Due to the delay imposed by the connection to the remote system, however, this would have a negligible effect. On the other side, system calls such as `splice_read`, that have been introduced for performance reasons, can use their regular counterparts (i.e. `read`) without breaking their basic functionality. With PROSPECT, we thus focus on *basic* character device operations that are broadly used by the majority of the character device drivers we analyzed. Specifically, PROSPECT can handle the `file_operations` listed in Table 6.3. The first column shows the name of the system call. For each system call, we specify if the system call is supported by PROSPECT, which system call is used to implement it and whether the call is handled locally or forwarded to the remote system.
With the exception of `mmap`, all operations that are frequently used for character device communication are supported. This is due to the fact that FUSE (Filesystem in Userspace) does not

Syscall	supported	implemented through	local/ remote
aio_fsync	yes	fsync	local
aio_read	yes	read	remote
aio_write	yes	write	remote
check_flags	no	-	-
compat_ioctl	yes	ioctl	remote
fallocate	no	-	-
fasync	yes	fsync	local
flock	no	-	-
flush	yes	fsync	local
fsync	yes	fsync	local
get_unmapped_area	no	-	-
ioctl	yes	ioctl	-
llseek	yes	llseek	remote
lock	no	-	-
mmap	no	-	-
open	yes	open	remote
poll	yes	poll	remote
read	yes	read	remote
readdir	no	-	-
readv	no	-	-
release	yes	release	remote
sendpage	no	-	-
setlease	no	-	-
splice_read	yes	read	remote
splice_write	yes	write	remote
unlocked_ioctl	yes	ioctl	remote
write	yes	write	remote
writev	no	-	-

Table 6.3: Basic `file_operations` supported by PROSPECT

support direct `mmap` calls for character devices at the moment (see Section 6.7). However, special consideration was necessary for the `ioctl` system call, as its exact behavior can frequently not be inferred prior to the actual execution.

IOCTL Mechanism and its Shortcomings

The IOCTL (I/O control) mechanism allows more flexibility in the communication with underlying device files. In general, there are two types of IOCTLs: Well-formed and unrestrictive IOCTLs. The difference is that well-formed IOCTLs have information about the type of the call encoded in the request number, whereas unrestricted IOCTLs do not provide this kind of information.

```
#define _IOC(dir,type,nr,size)    \
((dir  << _IOC_DIRSHIFT)|(type  << _IOC_TYPESHIFT)|\
 (nr   << _IOC_NRSHIFT) |(size  << _IOC_SIZESHIFT))
```

Listing 6.1: Encoding for well-formed IOCTLs

Listing 6.1 shows how information such as the direction (e.g. read, write or none/execute), the request number or the amount of data (size) are encoded in the `ioctl` request parameter. By decoding this parameter prior to the actual `ioctl` call, it is possible to determine the direction of the data transfer and whether the provided parameter is a constant or a pointer to memory. In this case, PROSPECT could forward the request to the target system.
However, for unrestricted IOCTLs and prior to the actual `ioctl` execution, PROSPECT would have no way to determine the direction of the data, how much data should be transferred and whether a provided parameter is supposed to be a pointer or not. Since unrestricted IOCTLs are commonly used for device driver communication, we had to address this issue in the design of PROSPECT. We solved the challenge by introducing a concept we denote *Dynamic Memory Tunneling* which is described in Section 6.4.

6.4 Implementation

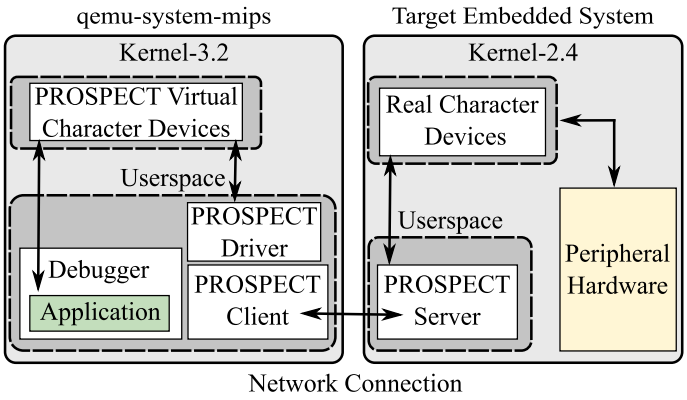

Figure 6.3: Peripheral Character Device Forwarding

As sketched in Sections 6.2 and 6.2, we assume that the analyst would like to dynamically instrument a binary on an embedded system that heavily accesses peripheral devices. While infeasible without PROSPECT, the application can now be executed inside an arbitrary analysis environment. Figure 6.3 provides a schematic overview of PROSPECT. On the left side, the application that should be analyzed is being executed within an arbitrary state-of-the-art debugger. However, instead of directly accessing the peripheral hardware through a character device, the application actually interacts with the *virtual* character devices that were generated through PROSPECT. At this point, PROSPECT intercepts the system calls defined in `file_operations` and forwards them to the userspace PROSPECT client. The client decides whether the system call should

be executed locally or on the remote system. If remote execution is required, it communicates with the lightweight PROSPECT server on the target system. The server has very low system requirements and can thus be executed on a wide range of embedded systems. Once the system call has been executed on the target system, any results are fed back into the software application under analysis.
In order to generate virtual character devices and intercept system calls, parts of PROSPECT need to run in kernel context. While we could have realized all parts of PROSPECT in kernel space, we decided that the major part of our implementation should be in user space. In comparison to a full kernel space implementation, a user space centric implementation has the advantage of increased system stability, security and, most of all, more flexibility. We implemented PROSPECT from scratch and our overall implementation consists of roughly $7,500$ lines of C code. In summary, PROSPECT comprises:

- A lightweight kernel driver utilizing the FUSE [91] kernel framework.
- A userspace driver combined with the PROSPECT client
- A lightweight server component running on the target system.

The PROSPECT lightweight server needs to be executed on the target system. Assuming that the security analyst typically has full physical access to the embedded device under test, we believe that this is a viable option. For instance, the analyst could use the bootloader console to get root level access to the operating system and then simply copy the PROSPECT statically linked binary to the device by using an attached storage medium or a networked remote machine as source.
Since our lightweight kernel driver utilizes the FUSE kernel framework, PROSPECT has the advantage that it is applicable to a wide range of operating systems, including Linux, FreeBSD, NetBSD, OpenSolaris, Android and OS X.

Concurrent Device Accesses

On typical embedded systems, a character device might be accessed by multiple threads or processes concurrently. Likewise, a single process or thread might interact with multiple devices at the same time. PROSPECT can handle these scenarios by using a client/server architecture (Figure 6.3) with multiple synchronization mechanisms. On the target system, there is a single PROSPECT server that can handle multiple incoming connections. Each client represents a character device that is forwarded to the target system, whereas each client can handle concurrent devices accesses by multiple threads and/or processes (Figure 6.4). PROSPECT relies on POSIX thread synchronization mechanisms (i.e. mutexes) to sustain the order of all accesses throughout the system.

File Descriptor Tracking

PROSPECT needs to keep track of file descriptors. Essentially, there are three cases we need to consider: (1) single processes, (2) child processes (i.e. created with `fork()`) and (3) threads. As file descriptors work on a per-process basis, they are only unique within the context of a process.

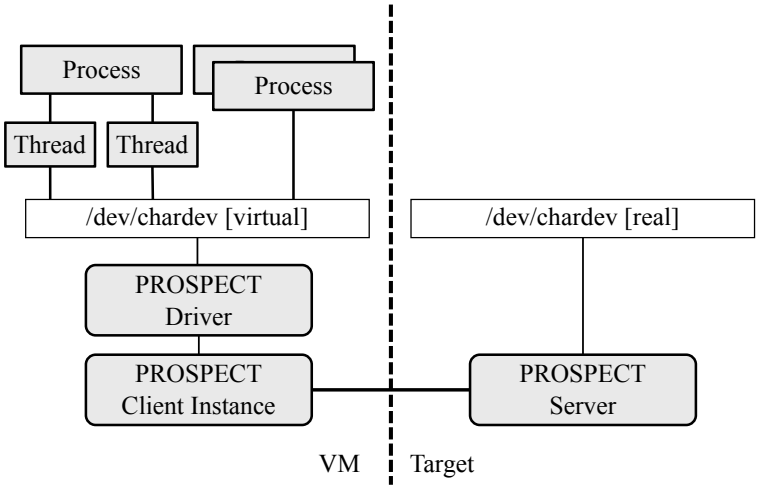

Figure 6.4: Concurrent Device Access

Whenever a new process is spawned, new file descriptors returned by `open()` typically start at 3 (as 0,1 and 2 are already used for *stdin*, *stdout* and *stderr*, respectively). Considering two different processes, both processes may receive the same file descriptor (i.e. 3), but it may correspond to completely different files with different properties (e.g. file offsets or permissions). If a process uses `fork()` to spawn a child process, it will inherit all open file descriptors from its parent, but any new file descriptors it receives at a later point will be unique to the child process. In contrast, threads behave much like a single process, as all file descriptors they receive are shared between them. As a result, both the PROSPECT client as well as the server would need to be aware of the type of process or thread in order to emulate normal operating system behavior.
PROSPECT tackles this challenge by taking a different approach. Instead of emulating the behavior of a real system, it uses *globally unique* file descriptors on the server side and supplies them in a synchronized way to all clients. More specifically, we implemented the PROSPECT server as a single process but with multiple threads to handle different connections. For that reason, all file descriptors it receives from the target's OS kernel are unique within the server and, ultimately, within all PROSPECT clients and all processes and/or threads accessing virtual character devices as well.

Dynamic Memory Tunneling

In Section 6.3, we explained how the IOCTL mechanism is used for more flexible device driver communication. However, unlike well-formed IOCTLs, their unrestricted counterparts do not provide any data exchange information (i.e. information on direction and the amount of data that should be exchanged with the driver). As unrestricted IOCTLs are frequently used, we considered different approaches to address this challenge in PROSPECT.
Both, the userspace application(s) accessing a character device as well as the character device driver are aware of the parameters for unrestricted IOCTLs. A userspace application may not use all unrestricted IOCTLs the driver supports. However, the driver implementation always includes

all supported unrestricted IOCTLs as well as the information on how data can be exchanged with them. Even better, device driver code is typically structured in a known way so that it can be loaded by the operating system. For that reason, we could extract the kernel image and device drivers from the target system and employ static (or even dynamic) code analysis techniques on the binaries to extract a data exchange rule-set for all available unrestricted IOCTLs. The drawback of this approach is that PROSPECT would need to be aware of operating system specifics (such as architecture, kernel version, kernel configuration, etc.). Thus, it would be hard to use PROSPECT on a wide range of different systems without major modifications. On the other side, extracting a rule-set from the userspace application, the analyst wants to work with, might be a challenge on its own (i.e. due to code size, program obfuscation or required manual code analysis).
Another approach we considered is that instead of extracting a rule set, PROSPECT could dynamically observe any unrestricted IOCTLs during program execution and learn from them. However, this is not always feasible, as the analysis would need to take place on the target system that does not necessarily support dynamic analysis in the first place. In fact, one of the goals of PROSPECT is to enable dynamic analysis on embedded systems, that might not support it for the reasons mentioned in Section 6.2.
Since any analysis required to gain information on unrestricted IOCTL parameters should not depend on the capabilities of the target system, we implemented *dynamic memory tunneling*. The key idea of dynamic memory tunneling is to always transfer a memory buffer to the target system if the IOCTL parameter is a possible pointer to a memory location. Accordingly, for each unrestricted IOCTL call, we need to answer the following questions:

- Is the parameter a valid pointer?
- How much data should be transferred to/from the target?

To determine whether the IOCTL parameter is a valid pointer, we use a heuristic. For each unrestricted IOCTL call, our system retrieves the PID (process ID) of the program that currently accesses the character device. For that PID it retrieves all mapped memory regions from the OS kernel (i.e. through `/proc/PID/maps`) and filters out any regions that are not suitable for a buffer (i.e. memory regions that are not read- and writable at the same time). If the parameter value is in one of the remaining memory regions, PROSPECT assumes that the parameter is a pointer and a data transfer with the target is initiated.
The question remains *how much* memory should be transferred to and from the target system. During our experiments we observed that the amount of data exchanged with unrestricted IOCTLs was below the page size (typically 4096 bytes on Linux) in all cases. To allow exceptions with larger buffer sizes, we experimentally limited the maximum size to $3{*}\text{PAGESIZE} = 12\text{KiB}$. However, PROSPECT can be easily reconfigured with increased limits. Besides the configured limit, the amount of memory that is actually transferred, can be limited through the mapped memory region boundaries as well.
During execution, for any unrestricted IOCTL call with a valid pointer as parameter, PROSPECT takes the following steps:

1. Given the pointer address ADDR and the PID of the calling process, use the kernel driver to read up to 3∗PAGESIZE bytes from the mapped memory region of the corresponding userspace process.
2. Transfer the buffer to the PROSPECT server on the target system and execute the unrestricted IOCTL call on a local copy of the transferred buffer.
3. Once `ioctl()` returns, compare the transferred buffer with the potentially modified local copy of the buffer to determine how many bytes were changed in the local copy.
4. In addition to the `ioctl()` *return* and *errno* values, send back the content of the local copy buffer to the corresponding client. The size of the transfer is limited through the last byte in the buffer that has actually changed (see Step 3).
5. Given the pointer address ADDR and the PID of the calling process, use the kernel driver to overwrite the corresponding memory region of the userspace process (i.e. starting at ADDR) with the content of the response buffer.
6. Return the `ioctl()` *return* and *errno* values to the calling process.

Through dynamic memory tunneling, PROSPECT can forward unrestricted IOCTLs with arbitrary read, write and execute operations.

6.5 Evaluation

To provide a well-founded discussion of our system, we evaluated PROSPECT in two ways. First, we collected system call timing information to determine the performance impact PROSPECT causes in comparison with the native system. Second, we conducted a case study over more than 6 months by running a full-scale security audit of a widely used commercial fire alarm system in the building automation domain.

Evaluation of Performance Impact

On a 324 MHz embedded Linux MIPS system with 16MiB RAM, we used the *strace* tool to collect timing information for the system calls that are used for basic character device access (see Table 6.3 in Section 6.3 for details). Table 6.4 shows the userspace system calls we monitored. To collect timing information, we ran a userspace application that makes heavy use of all of the system calls in Table 6.4. In order to determine how much longer the forwarded system calls take, we ran the application with PROSPECT in our analysis environment (qemu-system-mips) as well as natively on the embedded MIPS system. For both executions we used *strace* to create system call logs with timing information. For our measurements we collected timing information for $196,075$ system calls on the analysis environment and for $166,972$ system calls on the embedded MIPS system. To compare the execution time of the system calls, we created custom analysis scripts to keep track of the file descriptors. This way, we were able to consider only timing information for calls made on character devices that are forwarded when PROSPECT is used. The results are visible in Table 6.5 in Section 6.6.

Operation	Function
`close()`	Close device
`ioctl()`	I/O Control mechanism
`lseek()`	Seek to a given position
`_newselect()`	System call used for `poll()`
`open()`	Open device
`read()`	Read data from device
`write()`	Write data to device

Table 6.4: System Calls used for Character Device Access

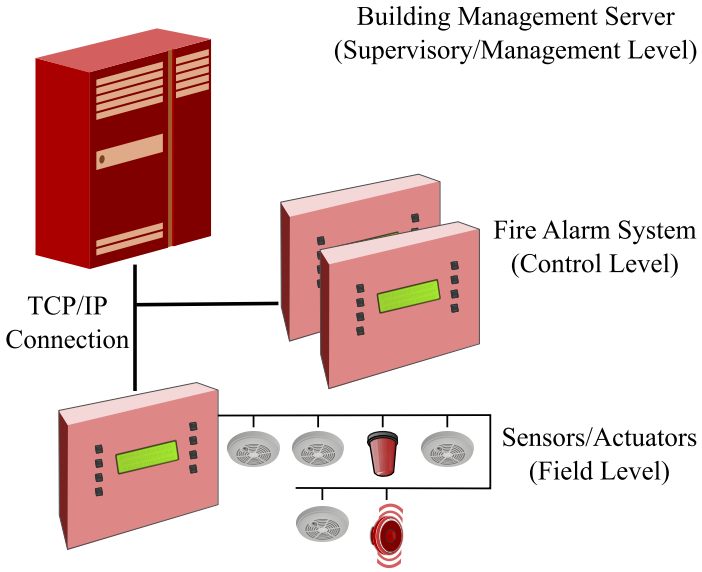

Figure 6.5: Proprietary Fire Alarm System

Case Study: Security Audit of a Proprietary Fire Alarm System

Under a legally binding non-disclosure agreement, we were able to employ PROSPECT to conduct a full-scale security audit of a widely used fire alarm system over a time frame of more than 6 months. A schematic overview of the overall fire alarm system is visible in Figure 6.5. On the lower side of the picture there is the fire alarm system that has a field level bus with a number of sensors (such as smoke detectors) and actuators (such as alarm lights or sirens) attached to it. Typically, there is one fire alarm system in a building and the sensors/actuators are situated in the rooms or on the outside of each building. Each fire alarm system is connected over a network connection (i.e. via TCP/IP) to one central building management server that is responsible for all fire alarm systems in multiple buildings. Thus, the server can manage the fire alarm systems and necessary steps can be taken in case there is a fire alarm. From the security perspective, the TCP/IP connection between the fire alarm systems and the building management server is

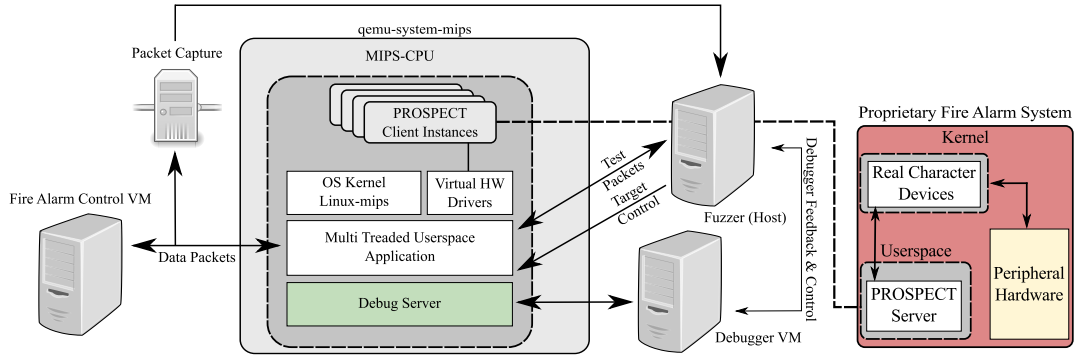

Figure 6.6: Security Analysis Environment

interesting. After all, if an attacker could get access to the fire alarm systems over the building network or even the Internet, it might be possible to trigger false alarms or to disable fire alarms which would lead to a dangerous situation for the persons in the building.

On a technical level, the fire alarm system we analyzed is a customized embedded Linux system with custom drivers, custom peripheral hardware components and several proprietary userspace programs that make up the overall fire alarm system implementation. The userspace programs make heavy use of multi-threading (pthreads) and the `fork` mechanism. In the running state, there is a total of 29 multi-threaded fire-alarm system specific processes, spawning multiple threads depending on the handled networking communication. In total, there are 5 different hardware peripherals that are accessed concurrently by the different processes and threads. As soon as the whole system is up and running, any network communication is processed. The system resources are very limited and the fire alarm implementation consumes nearly all available resources.

Due to the resource constraints, it is not possible to run a debugger on the system. Thus, dynamic analysis on the device is not possible either and the code that handles the network communication cannot be analyzed in another environment, as the device specific peripheral hardware would be missing there. As a result, the software application(s) would not start up in the first place. In this case, the analyst would be limited to static analysis and/or very basic security testing techniques.

To conduct a security audit of the fire alarm system implementation that handles the network communication, we employed PROSPECT to run the fire alarm system software implementation inside a virtual analysis environment. In this case, we utilized *qemu*, an open source virtualization environment that also supports the MIPS architecture. The center of Figure 6.6 shows the multi threaded userspace application with all connected installations for a complete analysis. It concurrently interacts with 5 different peripheral devices which are handled by multiple PROSPECT client instances, each handling exactly one character device. For automated fuzz testing of the network protocol implementation, we set up three different machines. On the left, there is the Fire Alarm Control VM that runs the manufacturer's software to communicate with the fire alarm system over a network connection. We used this machine to generate network traffic and capture it (*Packet Capture*) to obtain packets that can be used as input data for our fuzzer. Accordingly, the fuzzer can generate randomized traffic that looks very similar to the original

Syscall	Native [%]	Native [ms]	Fwd. [%]	Fwd. [ms]	Diff. [ms]	Slowdown [x]
`write()`	21.27	6.07	22.79	25.39	19.32	3.18
`lseek()`	28.14	0.12	18.88	2.31	2.2	18.65
`ioctl()`	1.6	0.89	4.27	117.15	116.26	130.37
`_newselect()`	14.8	43.27	17.14	40.85	-2.41	-0.06
`read()`	34.16	1.03	36.9	3.29	2.26	2.19
`close()`	0.01	0.1	0.0	N/A	N/A	N/A
`poll()`	0.0	N/A	0.0	N/A	N/A	N/A
`open()`	0.02	0.9	0.02	684.62	683.72	757.37

Table 6.5: PROSPECT Slowdown

communication protocol by taking packets from the captured network traffic, randomizing a single byte at a random position within the packet each test run and replaying the communication towards the userspace application under test. This allows us to use a single fuzzer implementation for a broad range of proprietary network protocols without the need to know protocol specifics or the requirement to develop a new fuzzer for each protocol. The downside of this approach is the limitation of the test cases to the captured network communication: If feasible protocol states are not captured during the capture phase, our fuzzer will not be able to test them. At the same time, we used a debug server to run the userspace applications we want to analyze. Through the Debugger VM (with a state-of-the-art debugger), the fuzzer can thus monitor the state of the software application and whether the test packets it sent, caused an exception such as a memory access violation. In this case, the fuzzer stores the network packets that led to the exception for later (manual) analysis.

6.6 Results and Discussion

Performance Impact

Table 6.5 shows the average performance impact of our system (Section 6.5). The values are arithmetic means over all recorded system calls. More specifically, the average number of `read()` accesses is the average of all read accesses from $166,972$ native and $196,075$ forwarded system calls, respectively. It clearly shows that for system calls that can be forwarded without further consideration (i.e. `lseek()`, `read()`, `write()` and `_newselect()`), the slowdown is practically insignificant as the main use of PROSPECT is program debugging (i.e. single stepping) and dynamic code analysis. Our results show that with PROSPECT the `_newselect()` call was slightly faster than on the native system. This is due to the nature of the system call. It blocks as long as either the given timeout is reached or one of the monitored file descriptors is ready. As this is closely related to the behavior of peripheral hardware (e.g. sleep modes), small variances in the recorded values are unavoidable.
In contrast, the `ioctl()` and `open()` calls cause a significant slowdown. The `ioctl()` slowdown is caused by the dynamic memory tunneling mechanism described in Section 6.4 whereas the `open()` slowdown is due to the connection establishment between the PROSPECT client and server. On the virtual analysis environment, we were unable to capture `close()` calls

on virtual character devices which is why we can not provide a performance comparison. However, as `close()` also works on an existing PROSPECT connection and no special considerations are necessary for the call, we believe that the performance impact is comparable to the `lseek()`, `read()`, `write()` and `_newselect()` calls. Furthermore, Table 6.5 also lists the frequency of each specific system call. It shows that the most frequently used calls are also the fastest. For instance, `write()`, `read()`, `seek()` and `_newselect()` account for 95,71% of all forwarded system calls. The distribution between system calls for native and forwarded execution slightly varied between analysis runs due to the internal state of the peripheral hardware.

Proprietary Fire Alarm System Security Audit

During our fire alarm security analysis (Section 6.5), we conducted extensive fuzz testing with the setup shown in Figure 6.6. During analysis, PROSPECT successfully forwarded more than $500,000$ system calls per analysis run to the target system without issues. Likewise, we were able to manually debug and single-step through the fire alarm application code. Our fuzz tests revealed a previously unknown zero-day vulnerability that was reported to the manufacturer. Our case study shows that even under demanding real-life requirements (29 multi-threaded processes that concurrently access 5 different hardware peripherals), our system performed well and enabled us to conduct both dynamic analysis and extensive fuzz testing to discover vulnerabilities.

6.7 Limitations and Future Work

Due to the nature of PROSPECT, it has a number of limitations that need to be considered. At the moment, our system uses TCP/IP over a network connection between the virtual analysis system and the embedded target system. However, if the userspace application under analysis changes the network configuration, this would also bring down the PROSPECT connection. Similarly, if the target system has no network interface, PROSPECT can not be used. We plan to add support for different communication interfaces (such as serial links) to PROSPECT so that it is usable in these cases as well. Another limitation is that PROSPECT requires *pthreads* on the target system and only runs on Linux right now. Another limitation is the missing `mmap` support for character devices due to the missing support in FUSE. Since `mmap` calls can be forwarded just the same (i.e. by using a similar approach as described in Section 6.4), we plan to implement full `mmap` support in future versions.

As PROSPECT requires only very little supported functionality on the target system and FUSE has been ported to a number of operating systems (Section 6.3), our system could be easily ported to different architectures and operating systems as well. At least for some implementations, the considerable slowdown caused by PROSPECT might lead to issues. However, this situation also occurs when single-stepping through programs and solutions, such as altering the information returned by timing related system calls, exist. Also, our system does not provide any security features at the moment.

Another consideration was briefly discussed in Section 6.3. When accessing devices on UNIX systems, their access rights are determined by the device's permissions. The client implementation needs to create virtual character devices and therefore requires root privileges.

In contrast, the PROSPECT server can be run as any user on the target system. It is however recommended to run it as root, simply to ensure that all devices are accessible. Through PROSPECT, the investigated process inherits the device access permissions from the server. As a result, it could be possible for an investigated process to access devices even though that would not be possible under normal circumstances. This property is not necessarily a limitation per se, as it constitutes an additional possibility to influence system behavior during analysis.
With regard to the high slowdown for unrestricted `ioctl()` calls, our implementation still provides room for improvement. For instance in future implementations, instead of querying the */proc* file system, we could implement a more efficient mechanism to minimize execution time.

6.8 Conclusion

PROSPECT turned out to be a valuable tool that enabled us to conduct a full-scale dynamic security analysis of a widely used fire alarm system. Without PROSPECT, dynamic analysis would have been infeasible due to the limitations of the fire alarm embedded system. We believe that PROSPECT's approach has a high practical impact. It allows to overcome the limitations of static analysis that are common for embedded vulnerability discovery. The general concept is applicable to a wide range of embedded systems, including smart phones or field level SCADA components.

6.9 Acknowledgements

The research leading to these results has received funding from the Austrian Research Promotion Agency (FFG) under grants 836276 (SG2), 834005 (Fire-IP) and the European Union Seventh Framework Programme under grant agreement n. 257007 (SysSec). We would like to thank the anonymous reviewers for their helpful feedback and improvement suggestions. We would like to thank Trustworks KG for providing valuable insights and tools that made this research possible.

CHAPTER 7

Embedded Security Testing with Peripheral Device Caching and Runtime Program State Approximation

The widespread use of embedded systems in security critical environments calls for better security testing techniques. However, testing embedded system firmware in its native environment imposes severe restrictions. Embedded systems can often be interfaced over debugging interfaces such as JTAG (Joint Test Action Group) or serial communication, but they typically only provide very basic debugging functionalities insufficient for more powerful security analysis techniques based on dynamic instrumentation. A possible solution to these problems is to create a VM (Virtual Machine) that emulates the entire embedded system. Since only the most common hardware is emulated by existing emulators, such as QEMU, real world embedded devices may require implementing additional peripheral device emulators. Yet, extending a VM with peripheral devices can not only be too time consuming for a resource constrained embedded security audit, but the information on the internals of these peripherals might not be available in the first place. Ultimately, this renders the emulation based approach infeasible in many cases. Previous work [56, 134] showed how peripheral devices can be transparently connected to a VM. This allows the embedded system firmware to run inside an emulator as if it were running on the original hardware with the peripheral devices directly attached. The extracted system firmware can thus be inspected outside its original system environment. The drawback of the peripheral device forwarding approach is the typically significant slowdown of device communication and the lack of possibilities to parallelize slow analysis runs or to leverage snapshots in presence of external peripheral device states. Since typical security testing techniques such as fuzz testing are highly repetitive in nature, in this work, we evaluate an approach utilizing caching of peripheral device communication in combination with runtime program state approximation. Our approach

could ultimately render existing dynamic firmware security analysis techniques more powerful by enabling functions such as snapshotting, test parallelization or testing without physical access to the embedded system. We show that the challenge is not the caching itself but the sufficiently accurate approximation of the embedded program state to decide which peripheral device response in the cache needs to be returned to the firmware under test. We address this problem with runtime program state approximation and show that, similar to symbolic execution, the approach suffers from state explosion. Specifically, the contributions presented in this paper are as follows:

- We present a peripheral device caching approach for embedded security testing.
- We present a state variable detection heuristic allowing runtime program state approximation as key to peripheral device communication caching.
- We evaluate the feasibility of our approach with programs from the GNU core utilities and show that it might be usable to address persistent drawbacks in embedded firmware security analysis in the future.

The remainder of this paper is organized as follows. Section 7.1 provides an overview of related work. In Section 7.2, we explain how peripheral devices are typically accessed from within an embedded operating system and describe why these devices are a challenge for current embedded system security testing methods. In Section 7.3, we present our peripheral caching approach leveraging runtime program state approximation which is described in Section 7.4. The results of our feasibility study are presented in Section 7.5. The conclusions and suggestions on further work can be found in Section 7.6.

7.1 State-of-the-Art and Related Work

In previous work, at least two different peripheral device forwarding approaches have been implemented. In [134], Zaddach et al. presented the Avatar framework allowing existing tools such as the QEMU emulator or symbolic execution tools to be connected to embedded target systems. Based on memory mappings, their system can forward peripheral device access from the emulator to the corresponding memory region of the peripheral device on the target embedded system. Similarly, Kammerstetter et al. presented the PROSPECT framework [56], an operating system centric approach that forwards peripheral device accesses from within the kernel in the VM to a stub on the embedded target device via a network connection. In addition to peripheral device communication forwarding, Koscher et al. presented SURROGATES [67], a system that uses Field Programmable Gate Arrays (FPGAs) to speed up the connection between the forwarding system (i.e., Avatar or PROSPECT) and the embedded hardware itself. In contrast, our work does not focus on the peripheral device forwarding techniques themselves, but instead adds a peripheral device communication caching layer in between the VM and the target device. We thus aim to simplify embedded security testing by enabling powerful mechanisms such as snapshotting, parallelization or testing without the analysis environment being connected to the real embedded system. The concept underlying our caching heuristic is related to the problem of program slicing where our peripheral caching system identifies states of the program slice that

deal with peripheral hardware access. In general, program slicing typically focuses on source code and has been broadly covered by Weiser et al. [124], Korel et al. [66], Frank Tip [110] and Binkley et al. [8]. More recently, Kiss et al. [61] and Cifuentes et al. [21] also covered the problem of slicing binary executables. Considering the work on binary slicing, our cache heuristic is loosely related as the cache needs to identify states in a program slice based on the runtime environment of the process. We thus aim at identifying individual states in a program slice of the running process without extracting the whole program slice.

7.2 Peripheral Device Access

By leveraging peripheral device forwarding, medium to large scale embedded systems can be analyzed for security vulnerabilities. Within this work, we exemplary focus on embedded systems that utilize Linux on a MIPS architecture such as routers or Cyber Physical System (CPS) components. These systems are typically composed of a System-on-Chip (SoC) containing a processor, ROM, SRAM and I/O Controllers. The I/O Controllers are used to connect the SoC to external components such as DRAM, flash memory or peripheral devices (see Figure 7.1). Depending on the embedded system use cases, connected external peripheral devices are often customly designed by system manufacturers and can range from simple sensors and actuators to complex modules such as communication interfaces or security modules.

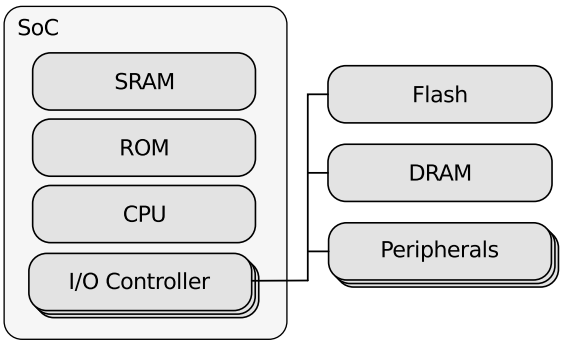

Figure 7.1: Typical Embedded System Hardware.

Challenges in Embedded System Security Testing

The security of embedded systems can be tested in several ways. The manufacturer of an embedded system typically has very detailed information about all components within the system and can thus resort to techniques such as whitebox security auditing and source code security analysis. Embedded systems often provide JTAG or serial console access allowing developers to access the running system. Depending on the specific implementation, these interfaces can provide a varying range of device access ranging from simple status readout to full dynamic system analysis. If the embedded system does not already provide tools for dynamic system analysis, the tester may be able to install necessary tools via an exposed debugging interface. However, embedded systems are typically resource constrained and tailored to a specific task.

Without the resources to run additional software like debuggers on the system, dynamic analysis on the device itself is often infeasible. In addition, the operating system kernel may be tailored to the specific use case of the system with debugging or system analysis features stripped to reduce hardware requirements and thus production costs. Whenever dynamic security analysis on the embedded system is not feasible, analysts typically aim at extracting the firmware from the device for further investigation. This can either involve static analysis techniques on the firmware with its well known limitations [71], as well as dynamic analysis approaches utilizing debugging interfaces such as JTAG or VM emulation. At this point, the challenge arises that embedded systems typically make extensive use of peripheral devices that are typically not available from within the VM. The analyst thus needs to resort to peripheral device forwarding frameworks such as Avatar or PROSPECT that have limitations on their own. Specifically, forwarding peripheral device communication is typically impeded by a significant slowdown and a lack of possibilities to parallelize slow analysis runs as each testing instance would require its own connected embedded target system.

Communication with Peripheral Devices

On UNIX systems such as Linux, peripheral devices are accessible via system calls that are handled by the kernel which in turn uses device specific hardware drivers for the actual device communication (Figure 7.2). In our test environment, the peripheral devices are represented as character devices. Depending on which commands the device driver supports, a user space program with the right permissions can thus access these devices with system calls such as `open`, `read`, `write` or `close`. To enable dynamic analysis on an otherwise resource constrained system, we utilize the PROSPECT framework [56]. The Linux kernel and all encompassing software running on the system are extracted from the embedded system and moved into an emulator such as QEMU.

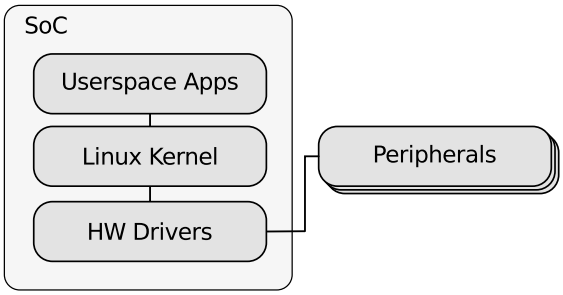

Figure 7.2: Typical Embedded System Software Stack.

To allow programs in the VM to communicate with the peripheral devices in a way similar to the original embedded system, PROSPECT replaces the embedded system software on the original hardware with a server stub that forwards all device communication over a network connection to the peripheral devices. We thus tunnel all device communication from the VM to the peripheral devices via a network connection such as TCP/IP over Ethernet. This allows us to run the embedded system software inside the analysis environment and thus enables the use

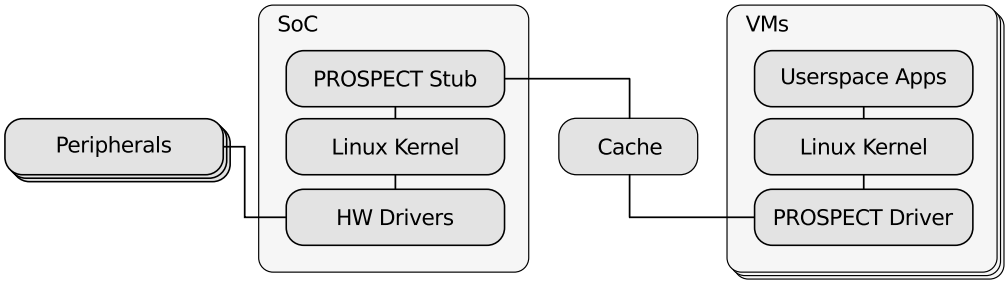

Figure 7.3: Embedded System Testing Utilizing PROSPECT with an Intermediate Cache.

of resource intense analysis techniques. Although the analysis environment typically provides significantly more system resources such as file system space, CPU speed or RAM, previous research showed that due to the peripheral device forwarding [56] most device communication will be significantly slowed down. Besides, another drawback is that each VM will require a dedicated set of embedded system hardware.

7.3 Caching Peripheral Device Communication

Considering security testing techniques such as fuzz testing, tests are typically highly repetitive and focused on very specific (i.e., security critical) code regions in the firmware. Triggered by each of those very similar test cases, the firmware of the embedded system performs the very same communication actions with its peripheral devices over and over again. For instance, consider a Real Time Clock (RTC) peripheral device that would be queried by the embedded firmware each time a network packet is received. Although the security analyst might only target the network packet handling code in the firmware with the fuzz tester, the peripheral device communication to the RTC would still need to be carried out as otherwise the firmware would stop to function and could not be tested. As long as the values returned from the RTC allow the firmware to continue its normal execution, it is not necessary that the returned values are actually correct. Although two subsequently read timestamps should represent an amount of time that has passed between the successive reads, the functionality of the firmware during the focused fuzz tests will in most cases not be impeded by the fact that the time itself is not correct. By adding a peripheral device communication cache between the analysis environment and the embedded system, the repetitive device communication actions could be stored so that during the highly similar test cases valid device responses can be served from the cache. Ultimately, this would enable very powerful supporting technologies such as snapshotting, parallel testing or even testing without the embedded system attached.

Caching Strategies

In the first step, we implemented a cache between the PROSPECT driver and its stub on the target device (Figure 7.3). For each peripheral device interaction, the cache receives the following information:

- Process Id (PID) and Thread Group Id (TGID)
- Name of the peripheral device
- Command type and command data

When the cache receives a command, it has to decide between two options:

1. Cache hit - An appropriate device response is already in the cache. The cached response is returned to the program without querying the actual device.

2. Cache miss - The cache has not stored a suitable device response. In this case, the cache first needs to bring the hardware into the state it would normally be before this request. This is done by resetting the hardware and replaying all communication that the requesting program performed until this point. The approach can thus forward the new command to the peripheral device, store the new reply and forward it to the VM. This means that the cache needs to retain information not just about the commands and their replies, but also about the previous command history for each VM.

The main challenge is to find a strategy that can be applied to decide whether for a specific firmware program state a valid peripheral device response is already in the cache. We explored several strategies and describe them in the following:

Choosing Responses by Command

Very simple devices may be cached by command. To do so, the device must either be stateless, or the device's state must be deducible from the command. For example, if the device is a simple switch that is only controlled by an open and close command, the cache does not need any information other than the command itself to react accordingly. For instance, whenever the cache receives a control request to turn on the switch it could just return the cached response confirming that the switch has been turned on. However, as soon as a single control command can return different responses this approach is no longer applicable. An example where this approach would not work is the above mentioned RTC module which would return a different timestamp value for every read command.

Choosing Responses by Command and Command History

An improved strategy is to store information about the previously issued commands to a peripheral device. Based on the command history, the cache can decide if a suitable device response is already in the cache or not. A simplified deterministic example could be a program that reads from a peripheral device representing an incrementing counter with an initial reset. After reset, the program would read continuously increasing counter values (i.e., 1, 2, 3, etc.) on each execution. The read command itself may look the same, but depending on which and how many commands were issued to the peripheral device before, the replies to each command need to be different for every call. If the cache can learn a sufficient amount of requests and replies from the first *training* execution, it can replay the answers every subsequent time the program is executed. The problem with this strategy is that even if the behavior of the peripheral device is deterministic, it becomes insufficient as soon as multiple threads access the same device. In this case, the thread scheduler will cause a different execution order of threads for the same input and the behavior from the perspective of the cache will no longer be deterministic. Since the cache would need to consider all possible thread execution orders to respond to future requests, the strategy quickly becomes ineffective with an increasing amount of program indeterminism. Listing 7.1 shows an example where two threads cause the mentioned problem by accessing a temperature sensor and a communication interface at the same time.

```
1  def Thread1():
2    while(True):
3      temp = readTemperature()
4      if (temp > max):
5        sendMessage("High temperature")
6      sleep(0.1)
7
8  def Thread2():
9    while(True):
10     statusMsg = getStatusMessage()
11     statusMsg += readTemperature()
12     sendMessage(status)
13     sleep(2)
```

Listing 7.1: Threading Example with Read-Loop.

Choosing Responses by Program State Approximation

A more advanced strategy is to find a heuristic to identify abstract program states reflecting the current position within the program flow. When program execution is started, the program typically makes use of resources such as the CPU or stack memory. We could thus derive a set of relevant CPU registers (i.e., the instruction register, the stack pointer, the general purpose registers) and use this information to determine in which state the program currently resides in. Whenever a peripheral device is accessed (e.g., with a `read` system call), we use the program state to determine whether there is already a known peripheral device response in the cache. If this is not the case, we forward the peripheral device communication from the analysis environment to the real system and cache the response for later use. However, considering typical program constructs such as a loop reading a temperature value (Listing 7.1), it is very likely that the

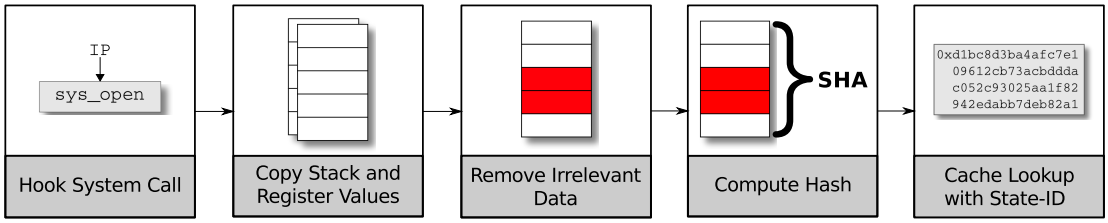

Figure 7.4: State Approximation Heuristic.

CPU registers will be identical within the `readTemperature()` function at the call site of the `read` system call for different loop iterations. It is thus necessary to include the program stack into the state computation so that the state of the outer function will be considered as well. However taking the stack memory into account, determining the program states gets much more challenging as it is no longer clear which memory regions are relevant with regard to the peripheral device communication. If the state approximation granularity is too low, many irrelevant memory regions will influence the program state approximation and different program states will be derived for the same peripheral device communication action (*state duplication*). As a result, most of the device accesses would be cache misses. In contrast, if the granularity is too high, we would get wrong cache hits and the program would receive invalid peripheral device responses. In the following, we present the runtime program state approximation approach we took and the results we were able to obtain with it.

7.4 Runtime Program State Approximation

On an Operating System (OS), the program state can be determined through its allocated memory (i.e., stack and heap), the CPU registers and handles received from the OS kernel (e.g., file handles). However, especially considering binary executables where the source code is not available, determining which variables need to be considered during the determination of the program state is considered to be a hard problem related to program slicing [21, 61]. Since it would not be feasible to deterministically detect exact program states, we implemented a heuristic (Figure 7.4) that attempts to approximate sufficiently exact program states to use them for our caching approach.

System Call Interception and Kernel/VM Hooking

In the first two steps of the heuristic (Figure 7.4), we need to intercept the systems calls used for peripheral device communication. For each intercepted system call, we need to decide whether the system call is utilized for communication with a device that is forwarded through PROSPECT. Furthermore, for runtime program state approximation we need to have access to the internals of the OS kernel and the program accessing the device as well. This includes the state of the virtual memory at the time of a call, the CPU registers and open file handles. We implemented and practically tested the following methods to obtain the required low-level information.

Virtual Machine Introspection (VMI)

The first method was implemented by extending QEMU with a Virtual Machine Introspection (VMI) module. VMI has the advantage that any low-level state information from the machine including physical memory or otherwise hard to access kernel internals can not only be accessed and read, but can be modified just the same. An additional advantage is that any introspection logic runs directly on the host machine and not inside the VM, leading to significantly higher performance. Although the VMI approach is very powerful, our VMI module implementation uncovered two major drawbacks. First, due to the low level VMI operates on, important functions inside the kernel such as those providing paging information and memory mapping need to be reimplemented. Even worse, important offsets to internal kernel structures can be configuration dependent requiring frequent adaptations of the VMI analysis code. Second, to reliably hook system calls, Translation Block Chaining (TBC) needs to be disabled. TBC is an optimization technique the QEMU emulator uses to drastically speed up emulation. Translation blocks are basic blocks of code from the guest system that are translated to the host system architecture. With TBC, these blocks are chained together and cached so that they do not have to be translated again each time the process counter arrives at that specific address. However, due to the caching, the program addresses within these cached blocks are no longer processed by QEMU's TBC lookup logic which ultimately causes our hooks on those addresses to no longer get executed. Disabling the TBC optimization allows reliable VMI hooking but at the same time significantly slows down the emulator.

Kernel Module

The second state approximation method was implemented as a loadable Linux kernel module running within the QEMU guest system. Since PROSPECT already performs system call hooking from within the kernel, we extended it with functions to read registers and mapped virtual memory regions of the calling process. Compared to the VMI approach, using a kernel module simplifies access to swapped out pages and kernel structures.

File Handles

Each time an `open` system call is used to return a new file handle, the value of the file handle is determined by the operating system kernel. Since the returned file handles frequently differ between executions, we use a file descriptor tracking mechanism. The mechanism places a hook on the `open` and `close` system calls. It can thus track the currently active file descriptors and remove them from the stack region of interest by overwriting the descriptors with zero bytes.

Registers

The register content has a central role in our program state variable detection heuristic. While the most important register to be utilized in this case is the instruction pointer, we found that the subset of registers leading to the best results also included the return address, the stack pointer and several general purpose registers.

Hash Computation and State-ID Matching

In the last two steps of the heuristic (Figure 7.4), we compute the SHA-256 digest and use it for cache lookup. The digest is computed on the concatenated stack region of interest and the register set. Using the previously described state variable detection heuristic, we ensure that hash digest results in a granularity that is suitable for cache lookups. The cache lookup is implemented as a large dictionary where the SHA-256 hash value is used as index to a device response data field of arbitrary size.

7.5 Results

Our feasibility study shows that our approach works for less complex programs but suffers from the well known state explosion problem for more complex programs. The low complexity programs we tested required less information from stack and registers to correctly determine the program state. However, with growing program complexity, it becomes more challenging to accurately determine a unique state suitable for cache lookups resulting in state duplication and cache misses. Since the number of these duplicates rises exponentially with increasing program complexity, similar to symbolic execution, the approach leads to the state explosion problem. In that regard, the MIPS architecture turned out to be especially challenging due to its standard calling convention and the resulting difficulty of stack frame unwinding. To test our approach, we used programs from the GNU core utilities and treated their file system accesses as peripheral device accesses with our caching approach in between. We tested 3 program classes:

1. Low Complexity Programs:
   For very simple programs such as `cat`, `head`, `sum` and `wc`, our caching approach hardly depends on stack frame information, no heap information is required and only a small subset of the registers is sufficient to correctly determine the program states for peripheral caching. Within a single execution the cache could thus already learn all necessary responses and use them correctly. At that point we were able to completely remove the program's input files and still obtain the identical program flow with our caching approach.

2. Medium Complexity Programs:
   Medium complexity programs such as `expand` rely on dynamic heap memory management. As a result, some of the relevant program states for device access may depend on the information stored at those memory regions. Using peripheral caching for programs like `expand`, the lack of information on heap content led to duplicate states. These could be compensated for by utilizing several training executions until the cache had learned all possible states including duplicates. It also required minor manual adaptations of the considered stack parameters within the heuristic. We believe that this problem can be addressed in future work and the heuristic could be greatly improved by adding proper stack unwinding. Monitoring the heap state would be an advantage, but is not mandatory. Without proper stack unwinding and manual adaptations, medium complexity programs currently present the limit of our approach.

3. Higher Complexity Programs
   Higher complexity programs such as `sort` not only heavily rely on dynamic heap memory management, but they also store a large amount of relevant state information on the heap. The problem and its possible solution are thus similar to medium complexity programs, but in comparison the number of duplicate states is much higher and can no longer be handled through manual adaptations. We believe that with stack unwinding and dynamic memory allocation monitoring the problem can be improved, but higher complexity programs will remain challenging.

7.6 Conclusion and Future Work

Our feasibility study showed that the presented peripheral caching concept could be an approach for some of the major drawbacks in embedded firmware security analysis. When applying typical embedded security testing techniques such as fuzz testing, sufficiently precise caching of peripheral device communication could thus enable powerful features such as snapshotting or test parallelization. After sufficient cache training the firmware can even be tested without requiring physical access to the embedded system. We showed that the problem is related to program slicing and may lead, similar to symbolic execution, to the well known state explosion problem. We created a VMI-based as well as a kernel-module based implementation and tested the feasibility of our approach with programs from the well known GNU core utilities package. Our results show that the peripheral caching approach works for low and medium complexity programs. However, depending on the architecture and the difficulty of stack frame unwinding, the program state approximation can become increasingly difficult. In future work, we're looking forward to port our approach to embedded architectures such as ARM allowing more precise stack unwinding. We believe that this will further increase the precision of the program state approximation so that more complex programs can be addressed with our approach as well. Furthermore, we aim to implement a kernel module/VMI hybrid implementation to benefit from the speed improvements of running the program state approximation heuristic outside the VM while still utilizing the OS kernel insight provided through a kernel module.

7.7 Acknowledgements

The research was funded by the Austrian Research Funding Agency's (FFG) KIRAS security research program through the $(SG)^2$ project under national FFG grant number 836276, the AnyPLACE project under EU H2020 grant number 646580, and IT security consulting company Trustworks KG.

CHAPTER 8

Efficient High-Speed WPA2 Brute Force Attacks using Scalable Low-Cost FPGA Clustering

Security in wireless Wi-Fi networks has come a long way. In comparison to wired network infrastructures, attackers are able to easily access Wi-Fi networks if they are in the vicinity. To protect Wi-Fi networks and the data being transfered over them, from the very beginning cryptographic protection mechanisms providing properties such as confidentiality, integrity or authenticity have been specified in the Wi-Fi IEEE 802.11 standard documents [46]. At the time WEP (Wired Equivalent Privacy) quickly turned out to be insecure allowing key recovery within minutes [9, 109], manufacturers started to implement several non-standard fixes such as WEP2 or WEPplus [68]. Ultimately, a switch-over to WPA (Wi-Fi Protected Access) employing the RC4-based TKIP (Temporal Key Integrity Protocol) as interim solution and to the longterm solution WPA2, in particular, has been suggested in the IEEE 802.11 standard documents [46]. Since 2012, WPA has been officially deprecated in the IEEE 802.11 standard and suffers from security vulnerabilities on its own [116]. In contrast, WPA2 is FIPS 140-2 compliant [114], much stronger and widely used to protect today's Wi-Fi infrastructures. The WPA2-Personal variant is designed for smaller networks and uses a pre-shared key (i.e., a Wi-Fi password) to derive the necessary key material for authentication, encryption and integrity protection. The Wi-Fi password needs to be at least 8 characters long and the key material is mainly derived through the state-of-the-art salted key derivation function PBKDF2 (Password-Based Key Derivation Function 2) [49] in combination with the SHA1 hashing algorithm [22] in HMAC configuration [23]. As a result, the security of a WPA2-Personal protected Wi-Fi network heavily relies on the quality of the password. Due to the computational complexity of the key derivation function and the use of the Wi-Fi's SSID as cryptographic salt, brute force attacks are very hard to conduct in the presence of random passwords with increasing length. Incurring significant costs well outside of what amateurs can afford, professional attackers can turn to commercial high-end FPGA-based

cluster solutions achieving WPA-2 password guessing speeds of 1 million guesses per second and more [89].
In this paper, we focus on the WPA2-Personal key derivation function and low-cost FPGA cluster based attacks that are not only affordable by professionals but by amateurs as well. Especially considering second-hand FPGA boards that have been used for cryptocurrency mining, those boards are now available to amateurs at low cost and can be repurposed to mount attacks on cryptographic systems. In the first part, we use a top-down approach to present WPA2-Personal security at a high level and we subsequently break it down to low-level SHA1 computations in high detail. In the second part, we use a bottom-up approach to show how these computations can be especially well addressed in hardware with FPGAs and we present how our solution can be integrated into a scalable low-cost system to conduct WPA-2 Personal brute force attacks. We evaluate our system with respect to performance and power usage, we compare it to results we obtained from GPUs and we conduct a real-world security evaluation case study showing the practical security impact of our system. Specifically, the contributions presented in this paper are as follows:

- We present a highly optimized design and architecture of a scalable and fully pipelined FPGA implementation for efficient WPA2 brute force attacks that brings the performance of today's highly expensive professional systems to the low-cost FPGA boards affordable by amateurs.
- Our implementation on Kintex-7 devices indicates that on the same hardware, our implementation is more than 5 times as fast in comparison to what is currently marketed to be world's fastest FPGA-based WPA2 password recovery system [32, 89].
- We implemented and evaluated our approach on three different low-cost FPGA architectures including an actual FPGA cluster comprising 36 Spartan 6 LX150T devices [129] located on a total of 9 second-hand repurposed cryptocurrency mining boards.
- We evaluate our system with respect to the power consumption and performance in comparison to GPU clusters, showing that FPGAs can achieve comparable or higher performance with considerably less power and space requirements, allowing attackers to create small and easy to use clusters.
- To highlight the practical real-world implications, we used the Wigle WiFi network dataset [126] to conduct a case study involving more than $166,988$ distinct Wi-Fi networks in 7 countries with potentially weak default passwords. Our results indicate that our system could be used to break into each of those networks requiring 3 days per network on our low-cost FPGA cluster in the worst case.

8.1 State-of-the-Art and Related Work

Since WPA2 is commonly used, there are several publications and projects dealing with WPA2 security and brute force attacks in particular. However, most of them rather focus on GPU brute force approaches and do not cover special purpose FPGA hardware, especially considering

low-cost FPGA hardware that is available to amateurs as well. For instance in [106], Visan covers typical CPU and GPU accelerated password recovery approaches with state-of-the-art tools like aircrack-ng[1] or Pyrit[2]. He considers a time-memory tradeoff usable for frequent Wi-Fi SSIDs and provides a performance overview of common GPUs and GPU cluster configurations. In that respect, oclHashcat[3] and the commercial Wireless Security Auditor software[4] need to be mentioned as well which are both password recovery frameworks with GPU acceleration and WPA2 support. Unlike these GPU-based approaches, our system comprises of a highly optimized and scalable FPGA implementation allowing higher performance at lower costs and power consumption in comparison. In [48], Johnson et al. present an FPGA architecture for the recovery of WPA and WPA2 keys. Although WPA support is mentioned, their implementation seems to support WPA2 only which is comparable to our system. However, while our implementation features multiple fully pipelined and heavily optimized cores for maximum performance, Johnson et al. only present a straight-forward sequential design leading to a significantly less performance in comparison. In [40], Güneysu et al. present the RIVYERA and COPACOBANA high-performance FPGA cluster systems for cryptanalysis. They provide details on exhaustive key search attacks for cryptographic algorithms such as DES, Hitag2 or Keeloq and have a larger cluster configuration than we had available for our tests. Yet, in contrast to our work, they do not cover WPA2 or exhaustive key search attacks on WPA2 in their work. As a result, it would be highly interesting to evaluate our FPGA implementation on their machines. Finally, Elcomsoft's commercial Distributed Password Recovery[5] software needs to be mentioned due to its support for WPA2 key recovery attacks on FPGA clusters [32, 89] and its claim to be world's fastest FPGA-based password cracking solution [31]. Although there is practically no publicly available information on the internals of their WPA2 implementation, in [89] performance data are provided. In contrast to their work, we do not only disclose our design, architecture and optimizations of our FPGA implementation, but we also claim that on the same professional FPGA hardware our implementation would be more than 5 times as fast. In comparison to the professional system, our system can achieve similar speeds on the low-cost repurposed cryptocurrently mining hardware that is available to many amateurs.

8.2 WPA2-Personal Handshake

Whenever a Wi-Fi client (Station) would like to connect to an Access Point, there are 802.11 management frames involved [46]. For instance in non-hidden Wi-Fi networks, the Access Point typically transmits beacon frames to advertise the network. In order to connect, a Station sends a probe request to determine network capabilities such as supported rates or vendor specific information. After that, in WPA2-Personal protected networks, Station and Access Point mutually authenticate against each other with the 4-way handshake depicted in Fig. 8.1.

[1] http://www.aircrack-ng.org
[2] https://code.google.com/p/pyrit
[3] http://hashcat.net/oclhashcat
[4] https://www.elcomsoft.com/ewsa.html
[5] https://www.elcomsoft.com/edpr.html

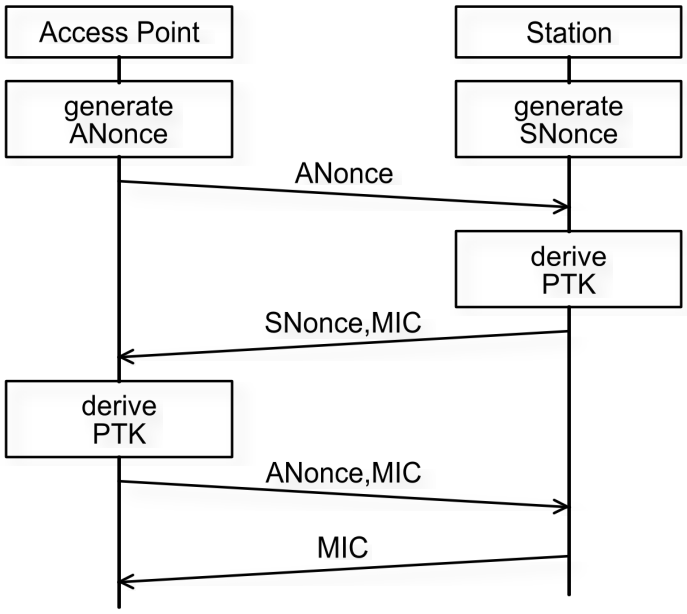

Figure 8.1: WPA2-Personal 4-Way Handshake

To start the mutual authentication process, the Access Point generates a 32 byte random `ANonce` and sends it to the Station. Similarly, the Station generates a 32 byte random `SNonce` and uses both nonces as well as the secret password to derive the PMK (Pairwise Master Key) and the Pairwise Transient Key (PTK) with the help of the WPA2-Personal key derivation functions described in the following Section 8.2. The nonces ensure that the handshake cannot by replayed by an attacker at a later time. Afterwards, the Station sends the `SNonce` back to the Access Point and utilizes the `PTK` truncated to the first 128 bits (denoted Key Confirmation Key - `KCK`) to compute a Message Integrity Code (MIC) over the packet data. At this point, the Access Point can already compare the received MIC with the computed one to validate that the Station is authentic and has knowledge of the secret password. In order to prove to the Station that the Access Point knows the secret password as well, the Station sends a message including `ANonce` and the corresponding `MIC` code. Since the Station can only compute the correct MIC code if it knows the necessary `PTK`, the Access Point can use this information for authentication. If the authentication was successful, the Station completes the handshake by sending a usually empty, but signed (`MIC`) message back to the Access Point. The Station can now associate to the Access Point and take part in the Wi-Fi network.

Key Derivation

In order to compute the `PTK` and its truncated variant (denoted the `KCK`) required to compute the `MIC` integrity code for provided packet data, the key derivation algorithm visible in Fig. 8.2 is utilized in WPA2-Personal. It uses the pre-shared secret key (i.e., the Wi-Fi network passphrase) and provided network information such as the SSID (i.e., the Wi-Fi network name), nonces and

the MAC addresses as inputs. To achieve a high level of security, at least two factors need to be considered in the key derivation. First, the key derivation algorithm needs to be collision resistant and computationally expensive. Collision resistant denotes the property that it is hard to find two different inputs that result in the same hash output when the hash function is applied. If the hash function is also computationally expensive, it will take an attacker longer to compute hash outputs thereby slowing down the number of guesses he can make per second. The longer and more complex the Wi-Fi password is, the more possible password combinations exist and the more hash computations the attacker needs to make to find the correct password. Second, the key derivation algorithm needs to be cryptographically salted so that, depending on the salt, different keys are generated for the same password. The general idea of salting is to add a random value to the message before the hash function is applied while the salt value is stored for later use. As a result, the same password will lead to different hash outputs since the salt value is different. Without the use of a salt, attackers could pre-compute lookup-tables for all possible passwords and corresponding hashes. While the size requirements of this table would grow tremendously with increasing password lengths, a practical time/memory tradeoff can be achieved with a pre-computed *rainbow table* [83]. The general idea of a rainbow table is to only store a small part of the possible hash value and password combinations. This is achieved by choosing a random password candidate as starting point and applying the hash function on it. However, instead of storing the hash value, the key idea is to define a reduction function that uses the hash value as input to create another valid password candidate. This process is continued to create entire *chains* where each chain ends either if the given length has been achieved or a password candidate has been created that is already a starting point for one of the already created chains. Since only the starting point and endpoint password candidates are stored, the storage requirements can be lowered. Once all possible chains have been pre-computed, the attacker can start to look up the password for a given hash value by computing the reduction function and locating the resulting password candidate in the stored endpoints of the chains in the rainbow table. As not all password candidates are stored in the table, it might very well be the case that the candidate can not be found. The attacker thus computes the hash output for the password candidate and applies the reduction function on the corresponding output to get another password candidate. This computationally expensive step is repeated until the password candidate is found among the endpoint password candidates of the chains in the table. Once found, the attacker takes the starting point password candidate and recomputes the intermediary values in the chain. At some point the computation will result in the hash value the attacker is looking for. The password will be the previously computed input value prior to this hashing step or the starting point password candidate itself (if it is the first hash value). Prior to computing a rainbow table, an attacker can thus freely choose the time/memory tradeoff between the required lookup time and the required storage space for the table. For more information, we would like to point to Martin Hellman's original paper [83].

To mitigate these threats, WPA2-Personal relies on the salted PBKDF2 [49] key derivation function. In the following, we describe the key derivation process in detail and closely focus on the required computational effort due to its impact on FPGA implementations and the achievable password guessing speed.

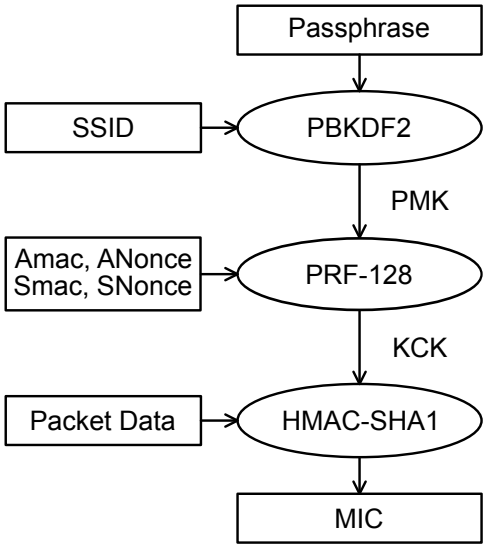

Figure 8.2: WPA2-Personal Key Derivation Function

Breaking it down to SHA1 Computations

Internally, the PBKDF2 key derivation function employed in WPA2-Personal utilizes $4,096$ iterations of the well known HMAC construction with the SHA1 cryptographic hash algorithm at its core to obtain 160 bit hash outputs (Fig. 8.3). Since the WPA2 Pairwise Master Key `PMK` needs to be 256 bits long, two PBKDF2 rounds are necessary. Their output is concatenated, but from the second iteration the output is truncated to 96 bits to achieve a 256 bit result. In both PBKDF2 iterations the secret password is used as key while the SSID of the Wi-Fi network concatenated with a 32 bit counter value serves as input. In the first iteration, the counter value is one while in the second iteration it is two. Consequently within both PBKDF2 iterations, there are $8,192$ HMAC-SHA1 iterations required to compute the `PMK` from the secret password and the network's SSID. With regard to the HMAC internals, Fig. 8.3 shows that a number of SHA1 iterations are necessary to obtain the MAC (Message Authentication Code). In general to compute the SHA1 hash digest of a message, the first SHA1 iteration is computed by using the initial SHA1 state and hashing the first part of the message. Depending on the length of the message, additional iterations might be necessary whereupon the previous SHA1 state output is used as state input for the next iteration. Once the full message has been hashed, SHA1 finalization needs to be applied by appending a '1' bit and the length of the message to the message itself and filling up the rest of the 512 bit SHA1 input block with '0' padding bytes. For WPA2-Personal key derivation, in the first PBKDF2 round the xor-transformation is applied on the password and the inner pad `ipad`. The result is a 512 bit block serving as input to the SHA1 hash function in initial state. The output is the HMAC inner state. Since the SSID may be no longer than 32 bytes, the hashing of the SSID and the 32 bit PBKDF2 round counter can be done together with the SHA1 finalization so that only one SHA1 iteration is necessary.

In the next step, the outer HMAC state is computed by hashing the xor of the password and the

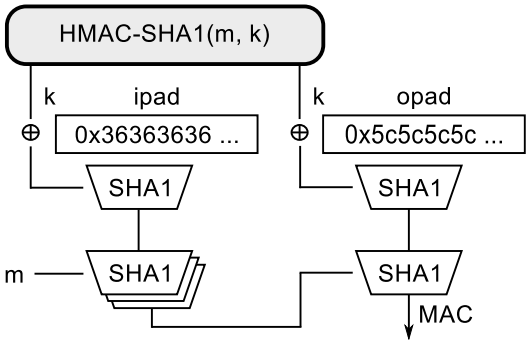

Figure 8.3: PBKDF2 core with SHA1 rounds in HMAC construction

outer pad `opad`. Afterwards, the previously finalized 160 bit digest is hashed and finalized with the outer state. At this point the MAC is ready. The second PBKDF2 iteration is computed in the same way with the difference that the round counter value is set to two instead of one. Since the password does not change during PBKDF2 iterations, the inner and outer HMAC states stay the same allowing us to use cached states instead of having to compute the states again. With that optimization in mind, it is required to compute at least $2 + 4,096 * 2$ SHA1 iterations for the first PBKDF2 round and $4,096 * 2$ SHA1 iterations for the second round (i.e., $16,386$ SHA1 iterations in total) to obtain the `PMK`. This computational effort, the use of the SSID as salt for key derivation and the security of the innermost SHA1 cryptographic hash function are three of the main reasons why WPA2-Personal key derivation is considered to be very strong against typical exhaustive key search attacks.
Once the `PMK` is available, the `KCK` is derived by applying a 128 bit Pseudo Random Function (PRF). Internally, it just uses HMAC-SHA1 again with the `PMK` as key. The hashed message is made up of the string "Pairwise key expansion", a terminating zero byte, an arithmetically sorted tuple of the Access Point and Station addresses as well as another sorted tuple of their nonces (i.e., `ANonce` and `SNonce`) including a finalizing zero byte. The `PTK` is the resulting MAC and it is truncated to the first 128 bits to obtain the `KCK`. If the `PMK` is available, the computation of the `KCK` takes 5 SHA1 iterations as due to the length of the `PMK` the finalization of the inner HMAC state can not be combined with the hashing of the `PMK`.
Whenever Access Point or Station would like to compute a `MIC`, they can do so by utilizing HMAC-SHA1 on the message with `KCK` as key. The result of the computation truncated to the first 128 bits is the `MIC`. The computational effort depends on the length of the message. However, considering the messages from the 4-way WPA2-Personal handshake, a total of 5 SHA1 iterations is required to compute the `MIC` since, similar to the `KCK` computation, the finalization of the inner HMAC state requires one additional iteration.

SHA1 Internals

Due to the high number of required SHA1 computations, it is essential to increase their speed as much as possible. To compute a SHA1 hash, a number of computational steps is necessary. Due to the high impact on our FPGA implementation, we provide a detailed overview of SHA1

internals. SHA1 [113] works on 512 bit chunks and produces a 160 bit hash digest when finished. If the message length is less than 512 bit, padding bits are used. For SHA1 finalization, a '1' bit, the padding bits (if necessary) and a 64 bit length field are appended. SHA1 has 80 internal rounds (denoted `t`) and requires a separate message working schedule W_t as well as a constant K_t for each of them. In the pre-processing step, the message working schedule W_t is computed as follows:

$$W_t = \begin{cases} M_t & 0 \leq t \leq 15 \\ rol(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}, 1) & 16 \leq t \leq 79 \end{cases}$$

The schedule $W_0 \ldots W_{15}$ is the message broken up into 16 words with 32 bit length each. For the remaining 64 words, message expansion is used by applying the xor-operation on previous schedules and rotating the result one time to the left. The constants K_t for the rounds comprise of a set of 4 words:

$$K_t = \begin{cases} \text{5a827999} & 0 \leq t \leq 19 \\ \text{6ed9eba1} & 20 \leq t \leq 39 \\ \text{8f1bbcdc} & 40 \leq t \leq 59 \\ \text{ca62c1d6} & 60 \leq t \leq 79 \end{cases}$$

After precomputation, the 80 SHA1 rounds are performed. Each round is based on the compression round visible in Fig. 8.4 and works on five 32 bit words denoted `A` to `E` where `rol n` denotes a rotate left by `n` operation and the $\boxplus$ operator denotes an unsigned 32 bit addition. In the initial iteration, a constant initialization vector `H0` to `H4` is used as input for `A` to `E`. The difference between the rounds is the function f_t defined as follows:

$$f_t = \begin{cases} (x \wedge y) \oplus (\neg x \wedge z) & 0 \leq t \leq 19 \\ x \oplus y \oplus z & 20 \leq t \leq 39 \\ (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & 40 \leq t \leq 59 \\ x \oplus y \oplus z & 60 \leq t \leq 79 \end{cases}$$

After each round, the resulting words `A` to `E` are fed back as input to the next round. Once all 80 rounds have been computed, the resulting words are added to the initialization vector `H0` to `H4` and the concatenated result is the resulting hash digest. Subsequent SHA1 computations are computed in the same way except that instead of the initialization vector the hash digest from the previous block is used.

Attacking the 4-Way Handshake

If an attacker would like to determine the secret WPA2-Personal password, a 4-way WPA2-Personal handshake between a Station and the Access Point needs to be obtained first. This can either be done passively or with the help of an active de-authentication attack where the attacker spoofs the source address of the Access Point and sends de-authentication frames to

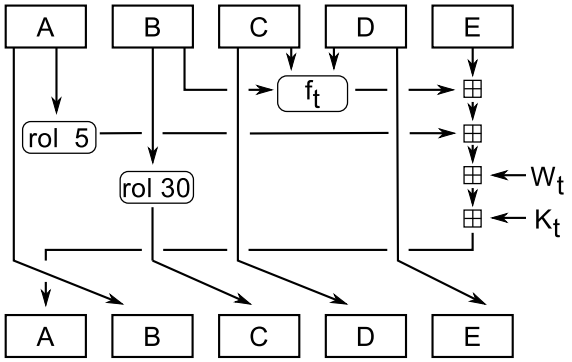

Figure 8.4: SHA1 Compression Round

the Station. Since those frames are not authenticated, the Station will falsely believe that the de-authentication request came from the genuine Access Point and will follow the request. However at a later time, it will re-authentication and thus give the attacker the opportunity to intercept the handshake. As soon as the attacker has the handshake, passwords can be guessed offline by deriving the key material for the `PMK` and the `KCK` and computing the `MIC` for one of the observed packets in the handshake. If the observed `MIC` is the same as the computed `MIC` for a password candidate, the attacker has found the correct secret password for the network. However, since a WPA2-Personal password needs to have a minimum length of 8 characters and for each password candidate a total of at least $16,386 + 5 + 5 = 16,396$ SHA1 iterations are necessary to compute the corresponding `MIC` over a handshake packet, exhaustive password guessing attacks are considered to be increasingly infeasible with higher password complexity and length. In the subsequent chapters, we show that the high computational effort can be addressed with special purpose FPGA hardware so that a high number of real-world WPA2-Personal protected networks with random passwords can be broken into within days.

8.3 FPGA Implementation

Implementing an algorithm on FPGAs is significantly different from software implementations. The FPGA comprises of different building blocks such as RAM or configurable logic blocks containing LUTs (Look Up Tables), Flip-Flops, dedicated arithmetic logic or shift registers. Initially, the inputs and outputs of all these building blocks are unconnected. During FPGA configuration, a bit-stream is uploaded to the configuration memory of the FPGA that subsequently sets up the interconnections using switch boxes.
Assuming at least some familiarity with the FPGA design, in the first step the designer utilizes a hardware description language (HDL) such as VHDL to describe the design. Based on the design, a synthesis tool creates a netlist that transfers the design to a set of interconnected high-level logic components. The netlist can thus be seen as a high-level schematic comprising the information which components are interconnected to each other through signals. In the next step, the components in the netlist are mapped to the building blocks of the specific targeted FPGA device such as LUTs, dedicated shift registers or memories. Afterwards, similar to the work

that needs to be done when designing a printed circuit board from a schematic, the components within the implementation need to be placed within the FPGA and the interconnects between those blocks need to be routed. The mapping, placing and routing steps are especially critical as often millions of interconnects need to be made and the signal run time for each of them needs to stay within specification. If only one signal requires a longer time from one register to the next one, the maximum clock frequency of the whole FPGA implementation will decrease to the clock frequency supported by the slowest path (i.e., the *critical path*). On the other hand the placement of the components is of paramount importance as well. If two components are placed non ideally and too far apart, their interconnects need not only be routed across a large area, but the time it takes for a signal to be transferred will also increase significantly thereby lowering the maximum clock speed of the entire implementation. Creating and especially optimizing high-speed FPGA implementations is thus highly challenging as apart from the logic design physical constraints such as the mapping, the signal routing or the electrical loads of signals need to be addressed as well. Since for large designs these steps often require multiple hours of design tool run time, performing optimizations can be hard as each design adaptation often requires another full design tool run. Only after the entire design flow has completed, the designer can get an impression whether the optimization was beneficial or not. However, the advantage is that depending on the requirements of an algorithm such as memory usage or the number of logic cells, it is often possible to achieve tremendous speedups, high scalability and lower power usage by efficiently using the resources of the FPGA and carefully optimizing the outcomes of the design steps from netlist generation to the final generation of the FPGA bit stream file. Considering the SHA1 internals described in Section 8.2, the algorithm is especially well suited for FPGA implementation due to the following reasons:

1. The algorithm has practically no memory requirements.

2. The rotate and shift operations utilized in SHA1 can be realized through FPGA interconnects with minimal time delay

3. Algebraic logic functions (xor, and, or, not, etc.) require minimal effort and can efficiently utilize the FPGAs LUTs

The most expensive operation are SHA1's additions due to the long carry chain between the adders. To implement the algorithm, a surrounding state machine is required to control which inputs should be supplied to the logic in different rounds. Considering that SHA1 has 80 rounds and we would like to achieve maximum performance, there are two design options: Either the SHA1 algorithm is implemented sequentially or in a fully pipelined way.
The advantage of a sequential implementation is that the FPGA can be completely filled up with relatively small SHA1 cores. However, the disadvantage is that each of those cores would require its own state machine which takes up a significant amount of space. In comparison, a fully pipelined implementation does not require an internal state machine as each of the SHA1 rounds is implemented in its own logic block. While this is a significant advantage enabling parallel processing, the drawback is that a fully pipelined implementation has much higher space and routing requirements. When using multiple cores (each containing a full pipeline), only an

integer number of cores can be placed so that a significant amount of unused space might be left on the FPGA. In our implementation, we also experimented with filling up this space with sequential cores but refrained from it due to the negative effect on the overall design complexity and the lower achievable clock speeds.
Due to the typically higher performance that can be achieved through pipelining and the property that we get one full SHA1 computation output per clock cycle per core, we targeted a heavily optimized and fully-pipelined approach. However, while pipelining alone has a considerable performance impact in comparison to a sequential approach, the key of obtaining maximum design performance are the optimizations.

FPGA Design

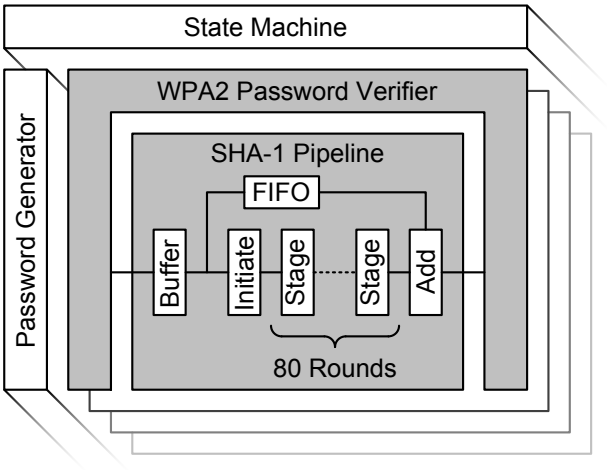

Figure 8.5: FPGA Design Overview

Our overall FPGA design is illustrated in Fig. 8.5 and has the following components: A shared password generator, a global brute force search state machine and an FPGA device specific number of brute force cores, each comprising a WPA2-Personal state machine with password verifier and a SHA1 pipeline.

Password Generator

The password generator (Fig. 8.6) is realized as a fast counter. Whenever the FPGA is idle, it can accept a new working block comprising of all necessary data including the actual start password (`start_password`) and how many passwords (`n`) should be tested. Initially starting at the start password, whenever the password generator is enabled (`enable`) it will output a new password (`current_password`) and the current password number (`count`) in each clock cycle. In case no more passwords can be fed into the brute force cores, the generator can be paused at any time by disabling the `enable` input. Ultimately, it will output new passwords until

n passwords have been reached and assert the done signal to indicate that all passwords within the current working block have been generated.

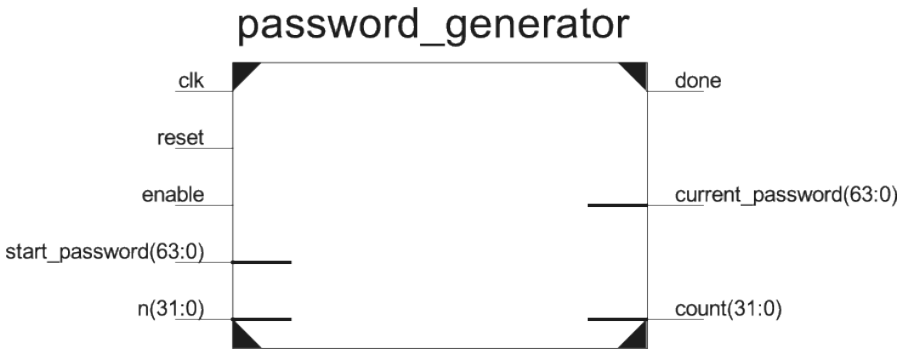

Figure 8.6: Password Generator Block

During the optimizations of our cryptographic cores in the design, at some point the long carry chain in the password counter became the clock speed limiting critical path. We were able to address the issue by parallelizing the counter and implementing the password carry with static multiplexers outside the sequential logic block. The sequential logic block can be seen as typical register transfer logic (RTL). With the clock signal, the old counter value is fetched from the source register, increased and finally output to the destination register. The path in between accounts for the delay. Since we need to have a carry overflow at the last valid password character (e.g., 'Z') we need a set of multiplexers that eventually reset the characters at each position of the password string. However, if this multiplexer based reset logic is within the sequential path it will also increase the time delay. By statically implementing the reset logic outside this sequential path we were able to balance the overall worst-case delays and achieved a password counter implementation that no longer accounted for the critical path in our overall design. Another password generator optimization approach we considered is utilizing multiple clock domains. The general idea is that the overall design naturally spends most of its time computing SHA1 iterations. At that time the password generator is disabled. We could thus use a less critical slower clock to generate the passwords and output them to clock synchronizing FIFO buffers directly placed next to the input of the SHA1 pipelines. As soon as a SHA1 pipeline requires a new password input, it can utilize its fast clock to drain the FIFO buffer which would in turn enable the password generator to refill the corresponding buffer at its slower clock. The advantages of this approach would be the following: First, the complexity of the password generator design can be further increased without negatively impacting the critical path. However second, the big advantage is the routing of the bus signals from the password generator to all the cores. Considering that the password generator is located at the center of the design and the passwords need to be distributed across the entire FPGA to all brute force cores, there is a significant impact on the time-driven routing complexity and the interconnect delays that negatively impact the maximum clock speed of the overall design. By leveraging a slower clock, the passwords would be already located in the FIFO buffers next to the SHA1 pipelines of each core but they could still be read with the fast clock the SHA1 pipelines are operating on. However, since with our previously mentioned password generator optimization the critical path was no longer within the

password generator domain, we did not implemented the approach. It will be covered in future work.

Global Brute Force State Machine

The task of the global brute force state machine is to constantly supply all brute force cores with new password candidates and check whether one of them found the correct password. Due to the insignificant speed impact and the advantage of lower design complexity we chose an iterative approach. Since our SHA1 pipeline comprises of 83 stages, we can concurrently test 83 passwords per brute force core. With our iterative approach, we enable the password generator and consecutively fill all brute force cores with passwords. Once all cores have been filled, the password generator is paused and we iteratively wait until all cores have completed. At that point, the password filling process is restarted. If a core finds the correct password or the password generator has reached the last password, the state machine jumps into the idle state and can accept the next working block. The penalty for this iterative approach is 83 clock cycles per core since once a brute force core has completed, we could immediately fill it with a new password. However, in comparison to the long run time of each core the impact is insignificant.

WPA2-Personal State Machine with Password Verifier

Each brute force core has a WPA-2 Personal state machine with a password verifier. It is the most complex state machine in the overall design. Its task is to compute the `MIC` code for each password candidate with the help of the SHA1 pipeline in its center. Each computed `MIC` is compared with the `MIC` from the WPA2-Personal 4-way handshake to determine whether the password candidate was correct or not. Figure 8.7 shows all necessary states and state transitions.

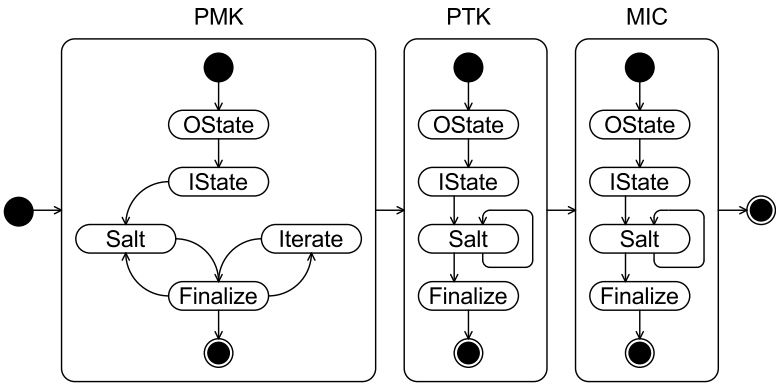

Figure 8.7: WPA2-Personal FPGA States

The state machine is divided into three WPA2-Personal key derivation phases: PMK computation (1), PTK computation (2), and MIC computation (3). The computation of the PMK has the highest computation effort due to the 2 PBKDF2 rounds with $4,096$ iterations requiring $16,386$ SHA1 iterations in total. Initially, 83 password candidates and the network's `SSID` are fed into the SHA1 pipeline to compute the corresponding HMAC outer and inner states (`OState`

and IState). Since these states do not change over the PBKDF2 iterations, the HMAC state computation needs to be done only once. In the first PBKDF2 round, the SSID and the PBKDF2 round counter (1) are used as salt. After that, there are $4,095$ more iterations in which the digest output is used as input. At that point, the second PBKDF2 round is computed by first computing the salt with an increased round counter value (2) and subsequently performing $4,095$ iterations to obtain the PMK.

SHA1 Pipeline

In each brute force core, the SHA1 pipeline occupies a large amount of space due to the high number of pipeline stages. While SHA1 has 80 rounds and a fully pipelined implementation would thus have an equal number of pipeline stages, we heavily optimized our pipeline to allow higher clock frequencies and consequently achieve more performance. The SHA1 pipeline is the key limiting factor of how fast our password guessing attacks can be conducted. Within the brute force cores, each of our SHA1 pipelines has 83 stages due to the optimizations we performed. Each core can thus compute 83 password candidates in parallel. The optimization approaches we applied are described in the following:
The first stage of the SHA1 pipeline is a buffer stage so that the delays of the different input logic blocks within the WPA-2 Personal state machine are not added to the pipeline's input logic and thereby does not increase the overall time delay of the critical path. The second stage denoted 'Initiate' is an optimization of the 4 required (expensive) additions in the E word of each SHA1 round. Instead of having all 4 additions in one stage, the structure of the SHA1 algorithm allows to pre-compute the output of the f function. The addition of the E word with the output of f and the key K_t enabled us to split up the required 4 sequential additions into two rounds with 2 additions, thereby significantly improving the maximum clock speed. Since the expansion steps for the message working schedule W_t require only a small amount of logic, another optimization is to do multiple message expansion steps in a single pipeline stage so that it is not needed in the following few stages. As a result, the source data is not accessed in each stage and shift register inference is boosted causing lower flip-flop fan-out as well as less power usage and lower area requirements. Another approach we took is the pipeline stage denoted 'Add' after the SHA1 rounds. After the last SHA1 round, the resulting digest is added either to the constant initialization vector H0 (first iteration) or to the previous digest for subsequent iterations. Due to these expensive additions, the design performance can be improved if they are carried out in a separate pipeline stage. Instead of forwarding the initial digest through all stages to the final addition stage, we leverage a FIFO-based delay line utilizing the FPGAs Block-RAM resources. This avoids excessive interconnect routing through all stages and thus makes the design smaller, reduces the number of critical paths and allows us to achieve higher clock frequencies more easily.

Additional FPGA Design Optimizations

In the WPA2-Personal state machine, we directly use the output from the password generator and compute the HMAC OState state first. At the same time, we store the password candidates in a Block-RAM buffer for later IState computation. After that, we no longer work with the

passwords but use password offsets instead. The result is a lower design density as no more additional interconnects are required for the password in later stages. A similar approach is used to avoid excessive interconnects and design density. Instead of having large buses, we either use Block-RAMs directly or form RAM-based delay lines to keep the `IState` and `OState` states as well as the computed `PMKs` and `PTKs` in memory. Instead of one large WPA2-Personal state multiplexer directly controlling all SHA1 pipeline inputs and outputs, we make use of several smaller and less complex multiplexers. Once again, this reduces overall design complexity and allows us to achieve higher clock speeds more easily. The top-level design needs to communicate with the outside world. Each time a new working block is added, all necessary Wi-Fi and WPA2-Personal data needs to be transferred and subsequently forwarded to all brute force cores. The result is a very broad bus spreading all over the FPGA design and causing severe design congestion. Since in our design only the password candidates and the `SSID` are required early within the WPA2-Personal state machine, we transfer the rest of the data over a small 16 bit bus leveraging inferred shift registers. This significantly reduces the complexity of the interconnects between the shared global state machine and the brute force cores across the FPGA. To lower the amount of input and output data exchanged with the outside world, we use a minimized Wi-Fi and WPA2-Personal data set that only includes the variable data fields from the captured handshake. All other data is not only fixed within the FPGA, but also kept locally in the cores. In addition, the FPGA does not output the correct password, but a numeric offset from the start password instead.

To avoid design congestion and to push the design to the highest clock speed possible, we make use of custom parameters within the Xilinx design tools for synthesis, mapping and routing such as the minimum inferred shift register size, register balancing or the number of cost tables. In addition, we use floor planning to support the mapper, placer and router in achieving higher clock rates. Floor planning is important to place critical components requiring a fast interconnect in between next to each other. In general, we were able to obtain the highest speed improvements by utilizing a star like topography: The password generator is distributed over the very center of the FPGA and the brute force cores are surrounding it. In addition we also used floor planning to avoid the placement of time critical components in FPGA areas that are hard to reach through interconnects. Especially considering the low cost Xilinx Spartan-6 and Artix-7 FPGA devices, we could identify major regions that can not be used to place components or interconnects. Consequently, we carefully placed critical components like the SHA1 pipelines in a way that those regions do not negatively impact the routing delay. In our FPGA implementations, we use a slow clock for communication with the outside world and a fast clock for computation at the same time. In our Spartan-6 implementation, the speed of the fast clock can be adjusted dynamically during runtime by programming the clock multiplier. In contrast, our Artix-7 implementation includes an automatic clock scaling mechanism to adjust the fast clock frequency with the device core temperature. Both approaches allow the FPGA design to run at high speeds without the danger of overheating.

Overall System Design

We implemented and practically evaluated our system on older model Xilinx Spartan-6 as well as on newer model Xilinx Artix-7 FPGAs. The Spartan-6 FPGAs are located on low-cost repurposed cryptocurrency mining boards. For comparison purposes, we created a full implementation for the more expensive Xilinx Kintex-7 XC7K410T FPGA as well, but could not practically test it since we did not have one of these FPGAs at hand. The overall system design for the Spartan-6 FPGAs is visible in Fig. 8.8 and based on ZTEX [135] FPGA boards. The Artix-7 design is similar but has only one XC7A200T FPGA on the board.

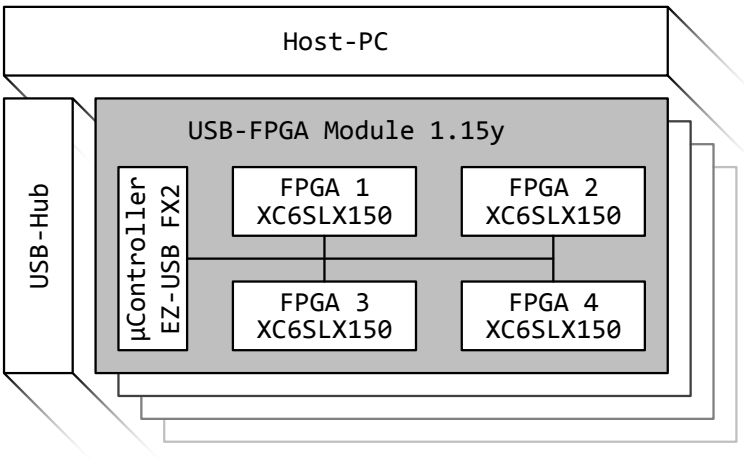

Figure 8.8: System Overview (Spartan 6 System)

The system comprises of a PC with a host software and several FPGA boards connected via the USB 2.0 high-speed interface. Each FPGA board has a fast EZ-USB FX2 micro-controller with custom firmware to interface with the FPGAs.

Our custom host software comprises ∼2k lines of Java code and utilizes the ZTEX SDK to allow easy communication with the micro-controller and the FPGAs. The host software accepts a configuration file that includes all necessary Wi-Fi and WPA-2 Personal handshake data. At startup, it enumerates all connected FPGA boards, uploads the micro-controller firmware if necessary and configures the FPGAs with our bit stream. The software makes use of several threads. Apart from the main program, there is a thread to generate password working blocks for the FPGAs and additional threads for each FPGA board. The password working blocks are kept in a pool with constant size. The device threads can supply working blocks to FPGAs and mark them as being processed. If an FPGA has finished a block, it is removed from the pool and the generator automatically creates a new working block. If for some reason an FPGA fails, the block sent to the FPGA is still in the pool and just needs to be unmarked so that the next free FPGA can process it instead.

The micro-controller firmware comprises ∼1k lines of C code and is responsible for USB communication with the host and communication with the FPGAs. Each FPGA has an 8 bit write and an 8 bit read bus in addition to read and write clocks, a write start control signal as well

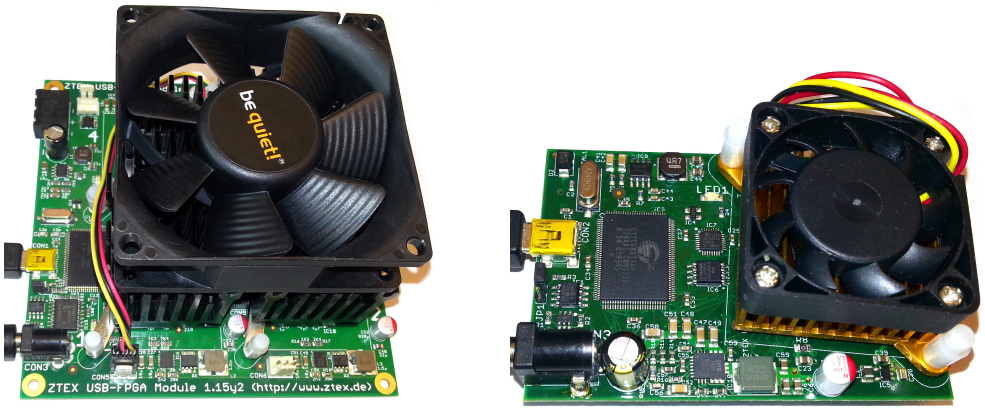

Figure 8.9: Ztex 1.15y Board (left), Ztex 2.16 Board (right)

as FPGA select signals and several programming signals to program the dynamic FPGA fast clock and the bit stream. Whenever the host software selects an FPGA on a board, the microcontroller asserts the corresponding select line in order to conduct subsequent bus communication or programming actions.

8.4 Evaluation

We performed multiple evaluations with regard to our design performance, the power usage and performance in comparison to GPUs as well as a Wi-Fi network security evaluation in the form of a case study.

FPGA Performance and Power Evaluation

We evaluated the performance and the power usage of our design on multiple FPGAs and FPGA boards. The first FPGA we targeted was an older model Spartan-6 XC6SLX150T-3 device. Four of these FPGAs can be found on the Ztex 1.15y board visible on the left side of Figure 8.9. The second FPGA we used for our evaluation was an Artix-7 XC7A200T-2 device on the Ztex 2.16 board visible on the right side of the picture. For both FPGAs, we created an optimized implementation and a configuration bit stream that can be uploaded to the device. The main difference between the bit streams is the FPGA type, the maximum clock frequency and most importantly the number of brute force cores we were able to fit onto the device.
To evaluate the performance and the power requirements, we used the obtained timing and power reports by utilizing the Xilinx timing and power analysis tools. In addition to these results, we also conducted practical measurements on the FPGA boards. At first, we measured the idle wattage of each unconfigured board at the power supply to determine the idle power usage. In the next step, we used a generated WPA2-Personal handshake with our software to mount a brute force attack on each of our FPGA boards. We used large password working packages resulting in a 30 seconds runtime per FPGA to avoid I/O bottlenecks. By measuring the wattage again during operation, we were able to determine the overall power consumption. To reduce the influence

of the power consumption caused by losses in the power supplies or components other than the FPGA, we obtained the power consumption of our FPGA implementation through the difference between the overall idle consumption and the consumption during operation. In Section 8.4, we use the same method to determine the power consumption of GPUs to get results that can be compared to the FPGA power consumption.
To obtain brute force performance measurements as well, we let each system run for at least 1 hour and computed the performance by measuring the number of password guesses during that time. The result is the average number of password guesses per second.
In addition to these evaluations, we executed the implementation on our FPGA cluster with 36 Spartan-6 XC6SLX150T FPGAs located on 9 Ztex 1.15y FPGA boards. The cluster setup allowed us to perform measurements on a larger setup and to determine how well our design scales with an increasing number of FPGAs. The setup is visible in Fig. 8.10 whereat two of those boards are not inside of the cluster as we use them for development purposes. During the tests, they were connected externally to the cluster. Using the power and performance measuring methodology from above, we obtained measurement results for the cluster as well.
To allow comparison with the commercial Elcomsoft WPA2-Personal FPGA cluster password recovery system [32, 89], we created an implementation and a configuration bitstream for the more expensive Kintex-7 XC7K410T-3 devices as well. However, since we did not have a board with this type of Kintex-7 FPGA, we can provide the Xilinx development tool's timing and power analysis results only.

GPU Comparison

To measure performance and power requirements of GPUs, we utilized cudaHashcat[6] v1.36 to mount brute force attacks on the same WPA2-Personal handshake we used previously to test our FPGA implementations. We executed the tool on machines with different Nvidia GPUs (GeForce GTX 750 Ti, GeForce GTX770 Windforce OC, GRID K520) and measured the performance in passwords per second as well as the power consumption. We applied the same power measurement methodology as during our FPGA evaluation. For the Amazon EC2 GPU cloud machines with GRID K520 GPUs, we were unable to obtain power measurements. The specific machine configurations and results are described in detail in Section 8.5.

Wi-Fi Security Evaluation - A Case Study

Driven by the high brute force speeds that can be achieved with FPGAs, we wanted to evaluate whether there is a real-world security impact. While long random passwords with a significantly large character set are practically infeasible to break within a reasonable time frame, the minimum WPA2-Personal password length is only 8 characters [46]. If the character set is limited as well, a random password can fall victim to brute force attacks within days or even hours if the brute force speed is high enough. To our surprise, discussions within our group suggested that the default WPA2-Personal passwords for many mobile Wi-Fi modems and even ISP provided modems/routers not only have a limited character set such as uppercase letters only, but the length

[6] http://hashcat.net/oclhashcat

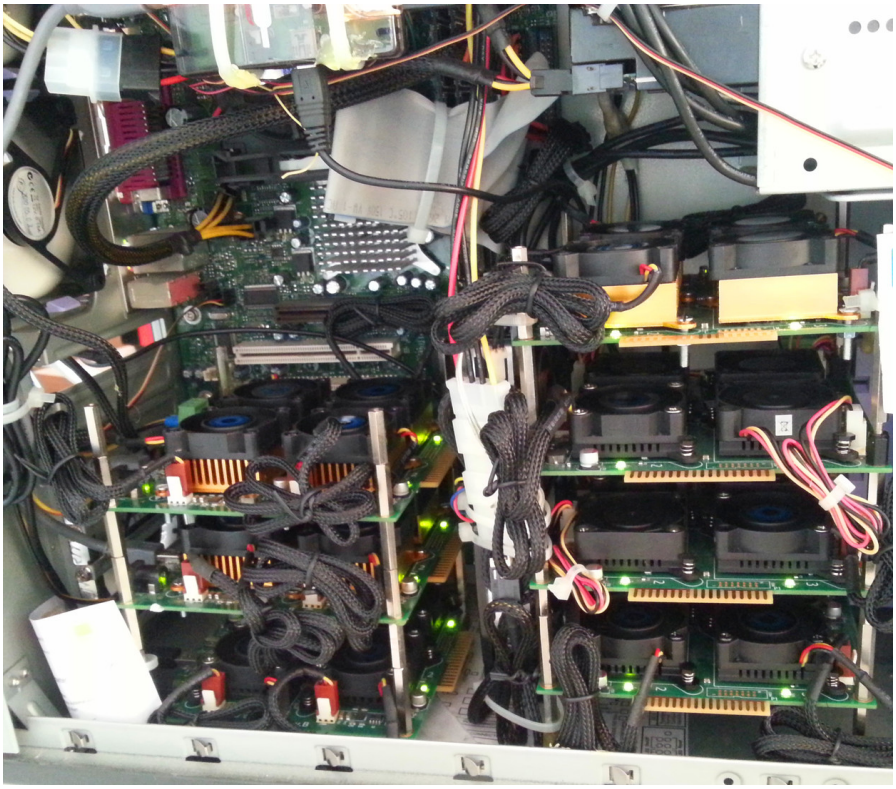

Figure 8.10: Spartan-6 XC6SLX150T Cluster

of the variable part of the password or the length of the password itself is also not more than 8 characters. Further investigation turned out that the largest ISP in our country uses weak default passwords for many of its Wi-Fi cable modems with only 8 characters length and comprising only uppercase letters. An example is provided in Fig. 8.11. The manual of the Wi-Fi enabled cable modem further confirmed our finding [112].

Figure 8.11: Bottom Side of a Cable Modem

While users can change these settings, the default `SSID` is `UPC<n>` where `n` denotes a number with either 6 or 7 digits. Under the assumption that most users also change the `SSID` of their network to something easier to remember when changing their Wi-Fi password, any visible `UPC<n>` Wi-Fi network would be an indication that the network is still using the weak default password.

To evaluate the practical security impact of our implementation, we used the Wigle war driving dataset [126] to get an approximation of how many of those likely to be insecure networks exist and whether those networks are limited to our country. To do so, we created a rectangular scanning grid across the country and queried the Wigle Web service [126]. Since the rectangular scanning grid also included parts of neighboring countries as well, it allowed us to see whether the same ISP is active in those countries and the potentially weak cable modems are used there as well. The results of our case study are presented in Section 8.5.

8.5 Results and Discussion

In the following, we present the results of the performance and power evaluation of our FPGA implementations, we present the obtained GPU WPA2-Personal brute force performance and power measurements results and comparison as well as the outcome of the Wi-Fi security evaluation case study.

FPGA Performance and Power Results

System	FPGAs	Type	Cost	Cores	Tool W	Tool MHz	Meas. W	Act. MHz	calc pwd/s	pwd/s	pwd/s W
Ztex 1.15y	1	XC6SLX150T-3	175	2	4.281	187	6.99*	180	21,956	21,871	3,128*
Ztex 1.15y	4	XC6SLX150T-3	700	8	17.124	187	27.96	180	87,826	87,461	3,128
9x Ztex 1.15y	36	XC6SLX150T-3	2,400	72	154.116	187	254	180	790,436	741,200	2,918
Ztex 2.16	1	XC7A200T-2	213	8	10.458	180	11.04	180	87,826	87,737	7,947
N/A	1	XC7K410T-3	2,248	16	25.634	216	N/A	N/A	210,783	N/A	N/A
N/A	48	XC7K410T-3	107,904	768	1,230.432	216	N/A	N/A	10,117,584	N/A	N/A

Table 8.1: Performance and Power Results of our Implementations for different FPGA Devices and Systems/Boards

The results for our FPGA performance and power evaluation are visible in Table 8.1. In the `System` and `FPGA` column the table shows on which systems we conducted our tests and how many FPGAs there are on the corresponding board and/or in the overall system. The FPGA device types are visible in the `Type` column whereat the name before the hyphen is the Xilinx device name and the number after the hyphen indicates the device speed grade (the higher the better). The `Cost` column provides an approximate cost estimate per FPGA in US$ we obtained by looking up the devices at common Xilinx distributors such as Digi-key[7]. However while the cost for 9 new Ztex 1.15y would be appoximately $6,300$ US$, we considered our 9 second-hand Ztex 1.15y boards previously used for cryptocurrency mining instead. We were able to obtain these boards for $2,400$ US$ which we believe is what amateurs could do as well, depending on how much boards they would like to acquire and how much they are willing to spend. The `Cores` column shows how many cores we were able to fit onto the device to achieve maximum performance. While more cores per device generally increase the performance, it can also cause the maximum clocking speed to drop significantly due to mapping, placement and routing issues. The table presents the implementations allowing us to achieve the maximum

[7] `http://www.digikey.com`

performance per device. The `Tool W` and `Tool MHz` columns present the design tool's power and timing analysis results. For the Spartan-6 FPGAs, we used the Xilinx ISE Suite 14.7 whereas for the newer 7-series devices Artix-7 and Kintex-7, we used Vivado Design Suite 2015.1. In general, it appeared that the newer Vivado tools produced better results, but since it doesn't support older model 6-series devices, we were unable to use it for our Spartan-6 implementations. The `Meas. W` and `Act. MHz` columns present the results for the power measurements we conducted on the FPGA boards/systems and the actual clock speed we used to run the devices. The `calc pwd/s` and `pwd/s` columns provide the WPA2-Personal performance in passwords per second whereas the first one indicates the calculated and theoretic maximum performance of our implementation whereas the latter one shows the actual measured average performance per board and/or system. In the last column `pwd/s W`, we use our actual power and performance measurements to determine how much brute force speed can be achieved per Watt which is especially important when scaling up our implementation to larger FPGA cluster systems. In the following, we discuss the results of our implementations on a per-device basis.

Spartan-6 Results

We used the Xilinx Spartan-6 XC6SLX150T-3 FPGA as the target for our initial implementation due to the availability of a high-performance FPGA cluster with 36 of these devices at our lab. The implementation on the Spartan-6 turned out to be especially challenging for multiple reasons. We had to deal with long design tool runs (3 hours of more) each time we made modifications to the design. Since the effects of many of our optimizations could not be tested through behavioral simulations alone, the duration of the design tool runs significantly slowed down the development. In addition, the internal switch boxes and types of slices in the Spartan-6 architecture are not well suited for more complex and larger implementations in comparison to newer 7-series devices. The result was that for many of our implementation attempts the device logic resources were sufficient, but the implementation still turned out to be unroutable due to the number of required interconnects. An important factor to achieve routable designs was our use of FPGA floor planning.
In summary, we were able to generate two implementations for the XC6SLX150T-3. One with 3 cores and one with 2 cores. While the first one has an additional core in comparison, it resulted in a much lower achievable clock speed (62.5 MHz) due to placing and routing issues effectively reducing the performance to that of a single core at high speed (180 MHz). In contrast, our optimized 2 core variant visible in Fig. 8.12 is able to run at up to 187 MHz leading to the highest performance we were able to achieve on the device. The picture shows the ready-to-upload placed and routed design. On the left and right the 2 brute force cores are clearly visible. In between the password generator and the global state machine are located. Although the dark areas indicate that there would be sufficient space for an additional core, our experiments showed that this would lead to lower performance as explained above.
The first 3 rows in Table 8.1 present the results we obtained through this implementation. Due to cooling requirements, we ran the design with a reduced clock speed of 180 MHz. Our measurements indicate that in this configuration, our implementation requires a total of 27.96W for all 4 FPGAs on the Ztex 1.15y board. The power measurements per Spartan-6 FPGA are marked with an asterisk to indicate that we were unable to measure them directly, but rather

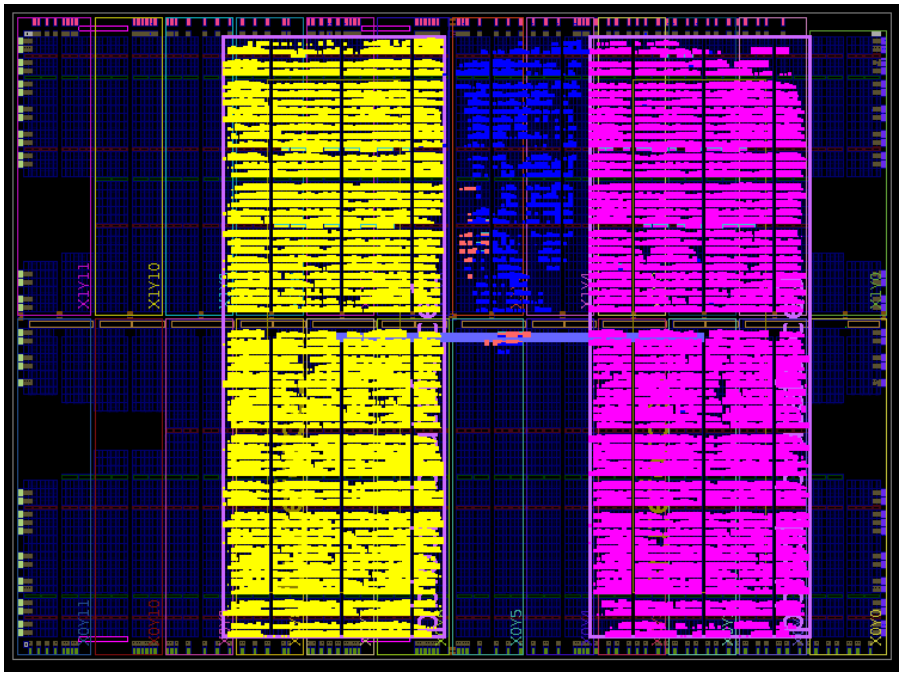

Figure 8.12: Placed and Routed XC6SLX150T

derived the measurement results from our power measurements for the entire Ztex 1.15y board with its 4 FPGAs. Our results show that our approach scales well and can be easily run in a cluster configuration producing a performance of 790, 436 password guesses per second on our cluster. The difference between the calculated maximum performance and the measured performance is mainly due to the I/O times between the PC, the microcontroller and the FPGAs. In addition, our Spartan-6 implementation includes a dynamic frequency scaling mechanism slowing down the FPGAs in case of device temperatures getting too high. With better cooling inside the cluster, we believe that the gap between the theoretic performance and the measured performance could be made smaller.

Artix-7 Results

In comparison to our Spartan-6 implementations, our implementations on the newer 7-series Artix-7 XC7A200T-2 FPGA required less effort as we could not only start from our already highly optimized Spartan-6 design, but the architecture and the newer Vivado design tool are also better suited for larger designs with increasing design complexity. Since device internals such as the clocks or PLLs are different from the Spartan-6 architecture, we had to adapt our implementation accordingly. The ability to read the device's core temperature from within the FPGA implementation was especially interesting. It allowed us to implement frequency scaling mechanisms directly on the FPGA not only preventing possible damage due to overheating, but also ensuring that each device always runs at the maximum performance possible. While we don't have access to an Artix-7 FPGA cluster, this feature would be especially helpful for

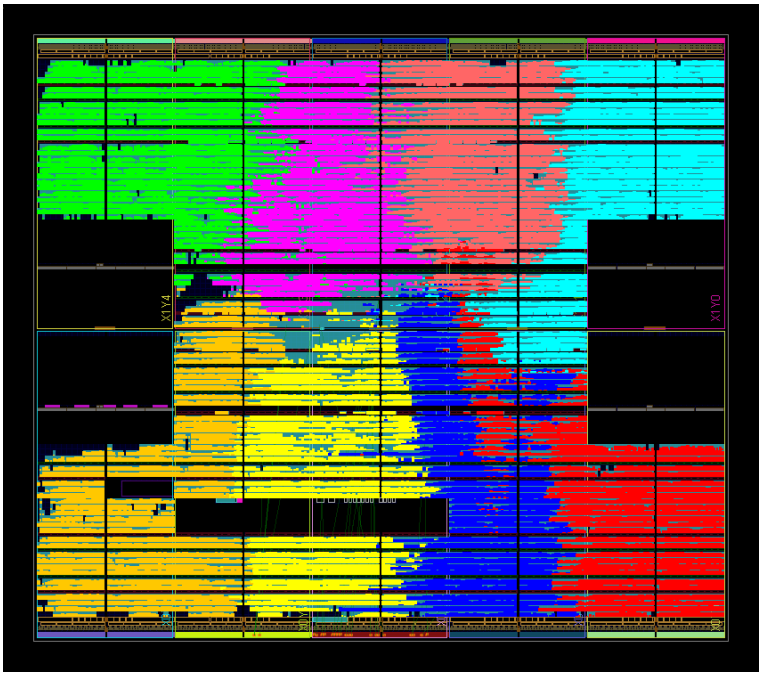

Figure 8.13: Placed and Routed XC7A200T

high-performance cluster designs.
Our ready-to-upload placed and routed design is visible in Fig. 8.13. The black blocks on the left and right are unusable areas. As routing around those areas makes it hard to meet timing constraints, we utilized floor planning to provide approximate locations for all of the 8 cores we managed to fit onto the device. All of the cores have a small path to the center where the small block with the global state machine and the password generator are located. The implementation can be run at up to 180 MHz to achieve a theoretic maximum of $87,826$ password guesses per second.
We managed to create an implementation with 9 cores as well, but similar to our Spartan-6 implementations the overall performance would have dropped due to the lower maximum clock frequency caused by placing and routing issues. With a measured performance of $87,737$ password guesses per second, our results show that a single XC7A200T-2 device achieves not only more performance than 4 of the older model Spartan-6 XC6SLX150T-3 FPGAs altogether, but it also requires just 11.04 Watt during operation.

Kintex-7 Results

In contrast to the low-cost Artix-7 FPGAs, Kintex-7 FPGAs are larger and allow higher performance but are also significantly more expensive. Although we didn't have any of those FPGAs at hand, we created an implementation for the Kintex-7 XC7K410T FPGA for two reasons.
First, Elcomsoft's marketed to be world's fastest FPGA-based WPA2 password recovery system relies on these FPGAs just the same and even provides performance figures for it [89]. Our target-

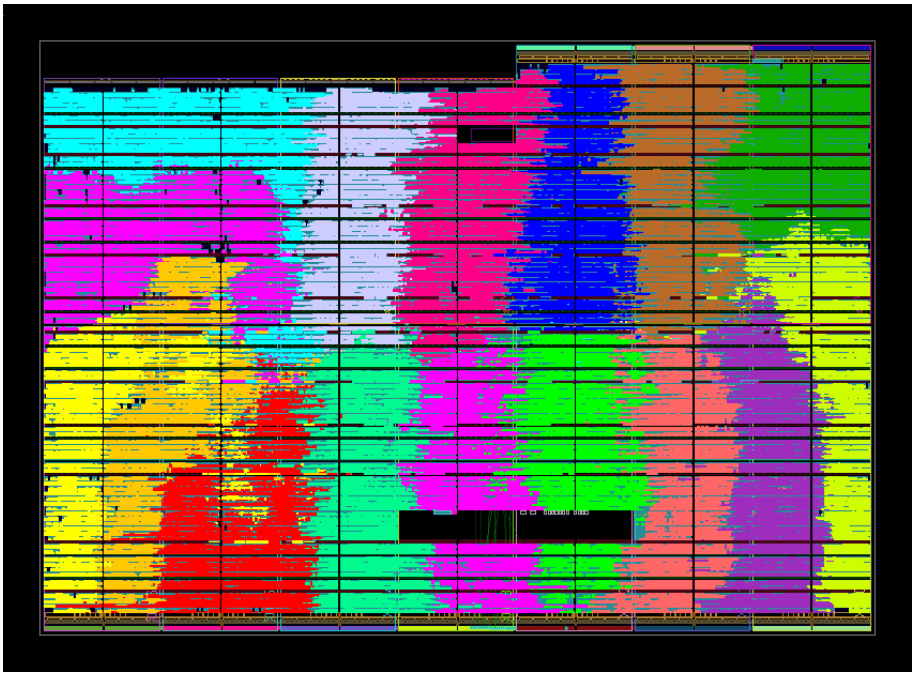

Figure 8.14: Placed and Routed XC7K410T

ing of the same FPGAs thus allows direct performance comparison between their implementation and ours. Their document indicates that on the PicoComputing SC5/M505-48 cluster with 48 XC7K410T FPGAs their implementation is able to produce $1,988,360$ passwords guesses per second [89]. Assuming that their implementation targets WPA2 employing SHA1 instead of WPA1 employing the much less complex MD5 algorithm, our implementation could achieve up to $10,117,584$ passwords per second on the same hardware and would thus be more than 5 times as fast.

Second, we wanted to obtain performance data for larger FPGAs as well. Although expensive, we believe that Kintex-7 FPGAs are well in the price range for professional attackers allowing them to achieve significantly more brute force attack performance per FPGA in comparison to low-cost FPGAs such as the Artix-7.

Our ready-to-upload placed and routed design is visible in Fig. 8.14. It comprises 16 cores running at up to 216 MHz. Similar to our Artix-7 implementation, the password generator and the global state machine are located in the center. However, due to the size and the thin layout of those units, they are hardly visible in the picture. At the same time, the image also suggests that with an increasing number of cores, the centralized state machine and password generator becomes a bottleneck due to the long bus interconnects reaching to the outside cores. We believe that this problem could be easily addressed by either using multiple shared state machines and password generators or by including FIFOs for the password candidates in each of the brute force cores. Due to the long runtime of each brute force core, the FIFO could be filled with a slow clock that can be easily routed across long distances on the FPGA. At the same time, the brute force cores would operate on the fast clock and drain the FIFOs. Due to their long run time, the

FIFOs could be easily re-filled through the slow clock before the next set of password candidates would be required.

GPU Results and Comparison

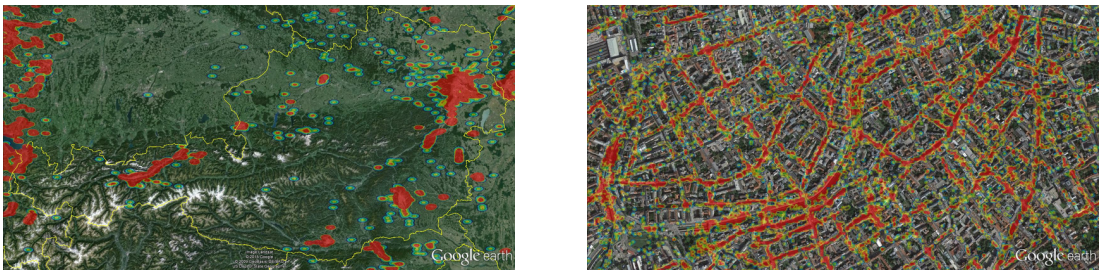

Figure 8.15: Density of `UPC<n>` networks with potentially weak WPA2-Personal Passwords

System	pwd/s	W	pwd/s W
GeForce GTX750 Ti	52,446	106	495
GeForce GTX770 OC	62,420	184	339
Amazon EC2 - GRID K520	30,370	N/A	N/A
Amazon EC2 - GRID K520 x4	109,073	N/A	N/A

Table 8.2: Performance and Power Results on GPUs

The results of our GPU evaluation (Section 8.4) are visible in Table 8.2. We performed the performance measurements by running cudaHashcat v1.36 on different systems and measuring the power consumption as the difference between idle and busy WPA2 computations to get results independent from other components in the system. The table shows the different GPU configurations (`System`) we used for our tests. The `pwd/s` column shows the performance in passwords per second and the `W` column indicates the power consumed by the GPU during runtime in Watt. The performance per Watt is visible in the `pwd/s W` column.
In addition to running GPU measurements on our own machines, we also conducted measurements on dedicated Amazon Elastic Cloud (EC2) GPU machines as well. While we could measure the performance on the machines just the same, we were unable to obtain power measurements. Although using a high number of GPU cloud machines appears promising to achieve high brute force attack performance, the limiting factor is the cost. Although our combined experiments on the dedicated Amazon EC2 machines took no longer than an hour, the costs we accumulated for our tests were already US$ 14.92. Since realistic brute force attacks might take considerably longer, the costs for an attacker would be far lower for acquiring a powerful GPU system instead of using the Amazon EC2 GPU nodes. In comparison to the results we obtained from our FPGA implementation, it is visible that GPUs can achieve the performance of a state-of-the-art low-cost FPGA (i.e., Artix-7), but their power consumption and performance per Watt is more than 10 times as high. At the same time, the performance achievable with a single larger FPGA such as

the Kintex-7 XC7K410T is no longer in the range of GPUs. Considering high-speed attacks with clusters, we believe that the scalability for FPGA-based attacks is better as well due to the small size of FPGAs, their lower power consumption and the high performance they can produce.

Wi-Fi Security Case Study

Area	Networks
Vienna	120,380
Austria + Border region	166,988

Table 8.3: UPC Networks with likely weak WPA2-Personal Passwords

The results for our real-world security evaluation case study are visible in Table 8.3. Within the city of Vienna, we found $120,380$ Wi-Fi networks with the `SSID` having the form `UPC<n>` in the Wigle [126] dataset. We were astonished by the high density of these networks within the city (Fig. 8.15 on the right), but assume that the real number of networks is even higher as not all networks are covered in the Wigle dataset. In addition, we discovered that the security issue is not only limited to the city of Vienna, but it is also persistent in the whole country and even in the border regions to neighboring countries. The left picture of Fig. 8.15 provides an overview of the network density. In total, we discovered $166,988$ of these networks within that area including neighboring countries.
Due to weak default configuration (Section 8.4), our case study suggests that our FPGA cluster implementation could be used to break into each of those networks in no more than 3 days considering the rate of $790,436$ guesses per second and the small number of only 26^8 password combinations for each vulnerable network. Running our implementation on the PicoComputing SC5/M505-48 cluster [89] instead, the necessary worst-case time to break a network would be further reduced down to 5.7 hours.
Due to the severe security implications and the high number of private networks involved, we already reported the problem to the ISP and are currently in the process of reporting it to the national CERT team as well.

8.6 Conclusion and Future Work

In this paper, we demonstrated that WPA2 passwords can be attacked at high speed rates not only by expensive professional FPGA cluster solutions but similar speeds can be achieved by amateurs on a budget as well, especially when considering second hand FPGA boards previously used for cryptocurrently mining. We specifically targeted low-cost FPGA devices, conducted implementations on 3 different FPGA architectures and evaluated our results with regard to performance and power. Our GPU evaluation suggests that FPGAs can not only achieve higher speeds at significantly less power, but they can also be used to easily create small and afforable FPGA clusters in the reach of amateurs. We conducted a real-world security evaluation showing that within the country and its border regions, there are more than $166,000$ Wi-Fi networks with

likely weak WPA2-Personal passwords that could be attacked through the implementation on our FPGA cluster within no more than 3 days each. Considering commercially used FPGA cluster systems, the time could be further reduced to no more than 5.7 hours depending on the cluster configuration and device types. However, we believe that besides the speedup we achieved it is more important to consider that the WPA2-Personal brute force performance achievable on professional systems is now becoming feasible on the low-cost systems amateurs can afford as well. We believe that these low-cost FPGA cluster based brute force attacks are thus a serious threat to real-world systems and need to be especially considered by manufacturers when choosing WPA2-Personal default passwords for hundred thousands of devices.
As counter measure, users need to increase the length of their passwords, the password should be random and it should utilize a large character set to increase password entropy.
In future work, we are looking forward to evaluate the security of other cryptographic systems as well. In that regard, we plan to design and implement a powerful low-cost FPGA cluster similar to COPACOBANA [40] but with low-cost 7-series devices instead.

Acknowledgments

The research was funded by the Austrian Research Funding Agency's (FFG) KIRAS security research program through the $(SG)^2$ project under national FFG grant number 836276, the AnyPLACE project under EU H2020 grant number 646580, and the IT security consulting company Trustworks KG who also provided the FPGA boards and the cluster.

Bibliography

[1] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. *Cryptographic Hardware and Embedded Systems - CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers*, chapter The EM Side—Channel(s), pages 29–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[2] AIT. SmartGrid Security Guidance. `http://www.ait.ac.at/research-services/research-services-safety-security/ict-security/reference-projects/sg2-smart-grid-security-guidance/`, 2013. [Online; accessed 14-October-2013].

[3] S. M. Amin and B. F. Wollenberg. Toward a smart grid: power delivery for the 21st century. *IEEE Power and Energy Magazine*, 3(5):34–41, September 2005.

[4] R. Araujo, E. Anjos, and D. Rousy Silva. Trends in the use of design thinking for embedded systems. In *Computational Science and Its Applications (ICCSA), 2015 15th International Conference on*, pages 82–86, June 2015.

[5] The IEEE Standards Association. IEEE Standard for Test Access Port and Boundary-Scan Architecture. *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001)*, pages 1–444, May 2013.

[6] A. Austin and L. Williams. One technique is not enough: A comparison of vulnerability discovery techniques. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*, pages 97–106, 2011.

[7] S. Bekrar, C. Bekrar, R. Groz, and L. Mounier. Finding software vulnerabilities by smart fuzzing. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, pages 427–430, 2011.

[8] David W Binkley and Keith Brian Gallagher. Program slicing. *Advances in Computers*, 43:1–50, 1996.

[9] Andrea Bittau, Mark Handley, and Joshua Lackey. The final nail in wep's coffin. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, SP '06, pages 386–400, Washington, DC, USA, 2006. IEEE Computer Society.

[10] Kaspersky Blog. Black Hat USA 2015: The full story of how that Jeep was hacked. https://blog.kaspersky.com/blackhat-jeep-cherokee-hack-explained/9493, 2015. [Online; retrieved 2016-03-18].

[11] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. *Advances in Cryptology — EUROCRYPT '97: International Conference on the Theory and Application of Cryptographic Techniques Konstanz, Germany, May 11–15, 1997 Proceedings*, chapter On the Importance of Checking Cryptographic Protocols for Faults, pages 37–51. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[12] David Brumley and Dan Boneh. Remote timing attacks are practical. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM'03, pages 1–1, Berkeley, CA, USA, 2003. USENIX Association.

[13] Dennis Brylow, Niels Damgaard, and Jens Palsberg. Static checking of interrupt-driven software. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 47–56, Washington, DC, USA, 2001. IEEE Computer Society.

[14] BSI. IT Baseline Protection Catalogs. http://www.bsi.bund.de/gshb, 2013.

[15] BSI. Protection Profile for the Gateway of a Smart Metering System. BSI-CC-PP-0073, 2013.

[16] BSI. Protection Profile for the Security Module of a Smart Metering System (Security Module PP). BSI-CC-PP-0077, 2013.

[17] Sang Kil Cha, T. Avgerinos, A. Rebert, and D. Brumley. Unleashing mayhem on binary code. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 380–394, 2012.

[18] D. Chang, M.T.-C. Lee, K. T Cheng, and M. Marek-Sadowska. Functional scan chain testing. In *Design, Automation and Test in Europe, 1998., Proceedings*, pages 278–283, Feb 1998.

[19] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. *Cryptographic Hardware and Embedded Systems - CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers*, chapter Template Attacks, pages 13–28. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[20] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco, 2011.

[21] Cristina Cifuentes and Mike Van Emmerik. Recovery of jump table case statements from binary code. In *Program Comprehension, 1999. Proceedings. Seventh International Workshop on*, pages 192–199. IEEE, 1999.

[22] D. Eastlake 3rd and P. Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174 (Informational), September 2001. Updated by RFCs 4634, 6234.

[23] D. Eastlake 3rd and T. Hansen. US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF). RFC 6234 (Informational), May 2011.

[24] J. Da Rolt, G. Di Natale, M.-L. Flottes, and B. Rouzeyre. New security threats against chips containing scan chain structures. In *Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on*, pages 110–110, June 2011.

[25] R. DeBlasio and C. Tom. Standards for the smart grid. In *IEEE Energy 2030 Conference*, pages 1–7, 2008.

[26] DRAS. The US Demand Response Automation Server. `http://drrc.lbl.gov/projects/draserver`, 2008. [Online; accessed 16-October-2013].

[27] Will Drewry and Tavis Ormandy. Flayer: Exposing application internals, 2007.

[28] E-Control. Risikoanalyse fuer die informationssysteme der elektrizitaetswirtschaft unter besonderer berücksichtigung von smart-metern und des datenschutzes. `https://www.e-control.at/documents/20903/-/-/3f89d470-7d5e-433c-b307-a6443692d8f7`, 2014. Accessed: 2016-04-01.

[29] E-Mobile Power Austria. Empora. `http://www.empora.eu/`, 2010. [Online; accessed 16-October-2013].

[30] EcoGrid. European FP7 Project. `http://www.opennode.eu`, 2012. [Online; accessed 16-October-2013].

[31] Elcomsoft. ElcomSoft and Pico Computing Demonstrate World's Fastest Password Cracking Solution. `https://www.elcomsoft.com/PR/Pico_120717_en.pdf`. [Online; accessed 13-Nov-2015].

[32] Elcomsoft Blog. Accelerating Password Recovery: the Addition of FPGA. `http://blog.elcomsoft.com/2012/07/accelerating-password-recovery-the-addition-of-fpga`, 2012. [Online; accessed 13-Nov-2015].

[33] ENISA. Appropriate security measures for smart grids. `http://www.enisa.europa.eu/activities/Resilience-and-CIIP/critical-infrastructure-and-services/smart-grids-and-smart-metering/appropriate-security-measures-for-smart-grids`, December 2012.

[34] European Union Agency for Network and Information Security. Appropriate security measures for smart grids. `https://www.enisa.europa.eu/activities/Resilience-and-CIIP/critical-infrastructure-and-services/smart-grids-and-smart-metering/appropriate-security-measures-for-smart-grids`, 2012. Accessed: 2015-12-02.

[35] Federal Office for Security in Information Technology. It baseline protection catalogs. https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/itgrundschutzkataloge_node.html.

[36] Federal Office for Security in Information Technology. Protection profile for the gateway of a smart metering system. https://www.commoncriteriaportal.org/files/ppfiles/pp0073b_pdf.pdf, 2014. Accessed: 2015-12-02.

[37] Federal Office for Security in Information Technology. Protection profile for the security module of a smart meter gateway. https://www.commoncriteriaportal.org/files/ppfiles/pp0077b_pdf.pdf, 2014. Accessed: 2015-12-02.

[38] Tal Garfinkel and Mendel Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *In Proc. Network and Distributed Systems Security Symposium*, pages 191–206, 2003.

[39] CEN-CENELEC-ETSI Smart Grid Coordination Group. Smart grid reference architecture. http://ec.europa.eu/energy/sites/ener/files/documents/xpert_group1_reference_architecture.pdf, 2012. Accessed: 2015-12-02.

[40] Tim Güneysu, Timo Kasper, Martin Novotný, Christof Paar, Lars Wienbrandt, and Ralf Zimmermann. High-Performance Cryptanalysis on RIVYERA and COPACOBANA Computing Systems. In Wim Vanderbauwhede and Khaled Benkrid, editors, *High-Performance Computing Using FPGAs*, pages 335–366. Springer New York, 2013.

[41] D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 129–142, May 2008.

[42] David Hely, Marie-Lise Flottes, Frederic Bancel, Bruno Rouzeyre, Nicolas Berard, and Michel Renovell. Scan Design and Secure Chip. In *Proceedings of the International On-Line Testing Symposium, 10th IEEE*, IOLTS '04, pages 219–, Washington, DC, USA, 2004. IEEE Computer Society.

[43] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1(4):293–302, 2011.

[44] Hui Hou, Jianzhong Zhou, Yongchuan Zhang, and Xiongkai He. A brief analysis on differences of risk assessment between smart grid and traditional power grid. In *Knowledge Acquisition and Modeling (KAM), 2011 Fourth International Symposium on*, pages 188–191, 2011.

[45] Andrew Huang. Keeping Secrets in Hardware: The Microsoft Xbox™; Case Study. In *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '02, pages 213–227, London, UK, UK, 2003. Springer-Verlag.

[46] IEEE-Inst. 802.11-2012 - IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Technical Report IEEE Std 802.11™-2012, IEEE-Inst, 2012.

[47] IEEE Standards Association. 1149.1-2013 - IEEE Standard for Test Access Port and Boundary-Scan Architecture, 2013.

[48] Tyler Johnson, Daniel Roggow, Phillip Jones, and Joseph Zambreno. An fpga architecture for the recovery of wpa/wpa2 keys. *Journal of Circuits, Systems, and Computers (JCSC)*, 24(7), 2015.

[49] B. Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898 (Informational), September 2000.

[50] Markus Kammerstetter, Daniel Burian, and Wolfgang Kastner. Embedded security testing with peripheral device caching and runtime program state approximation. In *Tenth International Conference on Emerging Security Information, Systems and Technologies (SECUWARE)*. Nice, France, 2016.

[51] Markus Kammerstetter, Lucie Langer, Florian Skopik, and Wolfgang Kastner. Architecture-driven smart grid security management. In *Proceedings of the 2Nd ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec '14, pages 153–158, New York, NY, USA, 2014. ACM.

[52] Markus Kammerstetter, Lucie Langer, Florian Skopik, Friederich Kupzog, and Wolfgang Kastner. Practical risk assessment using a cumulative smart grid model. In *Proceedings of the 3rd International Conference on Smart Grids and Green IT Systems*, pages 31–42, 2014.

[53] Markus Kammerstetter, Lucie Langer, Florian Skopik, Friederich Kupzog, and Wolfgang Kastner. Practical risk assessment using a cumulative smart grid model. In *3rd International Conference on Smart Grids and Green IT Systems (SMARTGREENS), April 3-4 2014, Barcelona, Spain*, 2014. To appear.

[54] Markus Kammerstetter, Markus Muellner, Daniel Burian, Christian Platzer, and Wolfgang Kastner. Breaking integrated circuit device security through test mode silicon reverse engineering. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 549–557, New York, NY, USA, 2014. ACM.

[55] Markus Kammerstetter, Markus Muellner, Christian Kudera, Daniel Burian, and Wolfgang Kastner. Efficient high-speed wpa2 brute force attacks using scalable low-cost fpga clustering. In *Conference on Cryptographic Hardware and Embedded Systems (CHES)*. Santa Barabara, CA, USA, 2016.

[56] Markus Kammerstetter, Christian Platzer, and Wolfgang Kastner. Prospect: Peripheral proxying supported embedded code testing. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '14, pages 329–340, New York, NY, USA, 2014. ACM.

[57] S. Karnouskos. Stuxnet worm impact on industrial cyber-physical system security. In *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, pages 4490–4494, 2011.

[58] M.M. Kermani, Meng Zhang, A. Raghunathan, and N.K. Jha. Emerging frontiers in embedded security. In *VLSI Design and 2013 12th International Conference on Embedded Systems (VLSID), 2013 26th International Conference on*, pages 203–208, 2013.

[59] Shubhangi Khare, Sandeep Saraswat, and Shrawan Kumar. Static program analysis of large embedded code base: An experience. In *Proceedings of the 4th India Software Engineering Conference*, ISEC '11, pages 99–102, New York, NY, USA, 2011. ACM.

[60] Himanshu Khurana, Mark Hadley, Ning Lu, and Deborah A. Frincke. Smart-grid security issues. *IEEE Security & Privacy*, 8(1):81–85, 2010.

[61] Akos Kiss, Judit Jász, Gábor Lehotai, and Tibor Gyimóthy. Interprocedural static slicing of binary executables. In *Source Code Analysis and Manipulation, 2003. Proceedings. Third IEEE International Workshop on*, pages 118–127. IEEE, 2003.

[62] Klimafonds. DG DemoNet - Smart LV Grid. `http://www.ait.ac.at/departments/energy/research-areas/electric-energy-infrastructure/smart-grids/dg-demonet-smart-lv-grid/`, 2012. [Online; accessed 16-October-2013].

[63] Paul C. Kocher. *Advances in Cryptology — CRYPTO '96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings*, chapter Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems, pages 104–113. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[64] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 388–397, London, UK, UK, 1999. Springer-Verlag.

[65] Philip Koopman. Embedded system security. *Computer*, 37(7):95–97, July 2004.

[66] Bogdan Korel and Janusz Laski. Dynamic program slicing. *Information Processing Letters*, 29(3):155–163, 1988.

[67] Karl Koscher, Tadayoshi Kohno, and David Molnar. Surrogates: Enabling near-real-time dynamic analyses of embedded systems. In *Proceedings of the 9th USENIX Conference on Offensive Technologies*, WOOT'15, pages 7–7, Berkeley, CA, USA, 2015. USENIX Association.

[68] A.H. Lashkari, M.M.S. Danesh, and B. Samadi. A survey on wireless security protocols (wep, wpa and wpa2/802.11i). In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*, pages 48–52, Aug 2009.

[69] Jeremy Lee, Mohammad Tehranipoor, and Jim Plusquellic. A Low-Cost Solution for Protecting IPs Against Scan-Based Side-Channel Attacks. In *Proceedings of the 24th IEEE VLSI Test Symposium*, VTS '06, pages 94–99, Washington, DC, USA, 2006. IEEE Computer Society.

[70] Bingchang Liu, Liang Shi, Zhuhua Cai, and Min Li. Software vulnerability discovery techniques: A survey. In *Multimedia Information Networking and Security (MINES), 2012 Fourth International Conference on*, pages 152–156, 2012.

[71] Bingchang Liu, Liang Shi, Zhuhua Cai, and Min Li. Software vulnerability discovery techniques: A survey. In *Multimedia Information Networking and Security (MINES), 2012 Fourth International Conference on*, pages 152–156. IEEE, 2012.

[72] Zhuo Lu, Xiang Lu, Wenye Wang, and C. Wang. Review and evaluation of security threats on the communication networks in the smart grid. In *MILITARY COMMUNICATIONS CONFERENCE, 2010 - MILCOM 2010*, pages 1830–1835, 2010.

[73] Wired magazine. Hackers Remotely Kill a Jeep on the Highway - With Me in It. `http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway`, 2015. [Online; retrieved 2016-03-18].

[74] Marcel Medwed and Elisabeth Oswald. *Information Security Applications: 9th International Workshop, WISA 2008, Jeju Island, Korea, September 23-25, 2008, Revised Selected Papers*, chapter Template Attacks on ECDSA, pages 14–27. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[75] Anthony R Metke and Randy L Ekl. Security technology for smart grid networks. *IEEE Transactions on Smart Grid*, 1(1):99–107, 2010.

[76] Charlie Miller and Chris Valasek. Remote Exploitation of an Unaltered Passenger Vehicle. `http://illmatics.com/Remote%20Car%20Hacking.pdf`, 2015. [Online; retrieved 2016-03-18].

[77] A. Mohan and H. Khurana. Towards addressing common security issues in smart grid specifications. In *Resilient Control Systems (ISRCS), 2012 5th International Symposium on*, pages 174–180, 2012.

[78] P. Moorthy and S.S. Bharathy. An efficient test pattern generator for high fault coverage in built-in-self-test applications. In *Computing, Communications and Networking Technologies (ICCCNT),2013 Fourth International Conference on*, pages 1–4, July 2013.

[79] Andreas Moser, Christopher Kruegel, and Engin Kirda. Exploring multiple execution paths for malware analysis. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 231–245, Washington, DC, USA, 2007. IEEE Computer Society.

[80] NIST. NIST Special Publication 1108R2 - NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 2.0, 2013.

[81] NIST. NISTIR 7628 - Guidelines for Smart Grid Cybersecurity, 2013.

[82] Karsten Nohl, David Evans, Starbug Starbug, and Henryk Plötz. Reverse-Engineering a Cryptographic RFID Tag. In *USENIX Security Symposium*, volume 28, 2008.

[83] Philippe Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, chapter 36, pages 617–630. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2003.

[84] National Institute of Standards and Technology. Introduction to nistir 7628 guidelines for smart grid cyber security. `http://www.nist.gov/smartgrid/upload/nistir-7628_total.pdf`, 2010. Accessed: 2015-12-02.

[85] National Institute of Standards and Technology. Framework and roadmap for smart grid interoperability standards, release 2.0. `http://www.nist.gov/smartgrid/upload/NIST_Framework_Release_2-0_corr.pdf`, 2012. Accessed: 2015-12-02.

[86] OGEMA. The German ICT Gateway Approach. `http://www.ogema.org`, 2012. [Online; accessed 16-October-2013].

[87] OpenNode. European FP7 Project. `http://www.opennode.eu`, 2012. [Online; accessed 16-October-2013].

[88] Sri Parameswaran and Tilman Wolf. Embedded systems security-an overview. *Design Automation for Embedded Systems*, 12(3):173–183, 2008.

[89] PicoComputing Inc. SC5-4U Overview. `http://picocomputing.com/brochures/SC5-4U.pdf`. [Online; accessed 13-Nov-2015].

[90] Ron Press. IC design-for-test and testability features. In *AUTOTESTCON, 2008 IEEE*, pages 88–91, Sept 2008.

[91] FUSE Project. Filesystem in userspace. *http://fuse.sourceforge.net/ (retrieved 2013-04-17)*, 2013.

[92] P.D. Ray, R. Harnoor, and M. Hentea. Smart power grid security: A unified risk management approach. In *Security Technology (ICCST), 2010 IEEE International Carnahan Conference on*, pages 276–285, 2010.

[93] Thilo Sauter, Stefan Soucek, Wolfgang Kastner, and Dietmar Dietrich. The evolution of factory and building automation. In *IEEE Magazine on Industrial Electronics*, pages 35–48, 2011.

[94] E.J. Schwartz, T. Avgerinos, and D. Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 317–331, 2010.

[95] Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitry Vyukov. Addresssanitizer: a fast address sanity checker. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference*, USENIX ATC'12, pages 28–28, Berkeley, CA, USA, 2012. USENIX Association.

[96] Zili Shao, Chun Xue, Qingfeng Zhuge, Meikang Qiu, Bin Xiao, and Edwin H.-M. Sha. Security protection and checking for embedded system integration against buffer overflow attacks via hardware/software. *IEEE Transactions on Computers*, 55(4):443–453, 2006.

[97] K V Shibu. *Introduction To Embedded Systems*. McGraw-Hill Education, 2009.

[98] Florian Skopik and Lucie Langer. Cyber security challenges in heterogeneous ict infrastructures of smart grids. *Journal of Communications*, 8(8):463–472, 2013.

[99] Florian Skopik and Paul Smith. *Smart Grid Security: Innovative Solutions for a Modernized Grid*. Elsevier Science Publishing, USA, 1th edition, 2015.

[100] Sergei Skorobogatov and Christopher Woods. Breakthrough Silicon Scanning Discovers Backdoor in Military Chip. In *Proceedings of the 14th International Conference on Cryptographic Hardware and Embedded Systems*, CHES'12, pages 23–40, Berlin, Heidelberg, 2012. Springer-Verlag.

[101] Sergei P. Skorobogatov and Ross J. Anderson. *Cryptographic Hardware and Embedded Systems - CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers*, chapter Optical Fault Induction Attacks, pages 2–12. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[102] Smart Grid Coordination Group, CEN-CENELEC-ETSI. Reports in response to smart grid mandate m/490. `http://www.cencenelec.eu/standards/sectors/SmartGrids/Pages/default.aspx`, 2012. [Online; accessed 16-October-2013].

[103] Smart Grid Coordination Group, CEN-CENELEC-ETSI. Smart grid reference architecture. `http://ec.europa.eu/energy/gas_electricity/smartgrids/doc/xpert_group1_reference_architecture.pdf`, 2012. [Online; accessed 15-October-2013].

[104] Smart Grid Modellregion Salzburg. ZUQDE. `http://www.smartgridssalzburg.at/forschungsfelder/stromnetze/zuqde/`, 2010. [Online; accessed 16-October-2013].

[105] Smart Grid Modellregion Salzburg. Smart Web Grid. `http://www.smartgridssalzburg.at/forschungsfelder/ikt/smart-web-grid/`, 2011. [Online; accessed 16-October-2013].

[106] Sorin Visan. WPA/WPA2 Password Security Testing using Graphics Processing Units. *Journal of Mobile, Embedded and Distributed Systems*, 5(4), 2013.

[107] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 635–647, New York, NY, USA, 2009. ACM.

[108] S. Sumpf and J. Brakensiek. Device driver isolation within virtualized embedded platforms. In *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, pages 1–5, 2009.

[109] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. Breaking 104 bit wep in less than 60 seconds. In Sehun Kim, Moti Yung, and Hyung-Woo Lee, editors, *Information Security Applications*, volume 4867 of *Lecture Notes in Computer Science*, pages 188–202. Springer Berlin Heidelberg, 2007.

[110] Frank Tip. A survey of program slicing techniques. *Journal of programming languages*, 3(3):121–189, 1995.

[111] L. H. Tsoukalas and R. Gao. From smart grids to an energy internet: Assumptions, architectures and requirements. In *DRPT*, pages 94–98, 2008.

[112] Ubee. Ubee EVW3226 Advanced Wireless Voice Gateway - Bedienungsanleitung. `http://www.unitymedia.de/content/dam/unitymedia-de/hilfe--service/doc/ubee-evw3226-installationsanleitung.pdf`, 2011. [Online; accessed 22-April-2015].

[113] U.S.Department of Commerce National Institute of Standards and Technology. FIPS PUB 180-2, Secure Hash Standard (SHS), 2002. U.S.Department of Commerce/National Institute of Standards and Technology.

[114] U.S.Department of Commerce/National Institute of Standards and Technology. FIPS PUB 140-2, Security Requirements for Cryptographic Modules, 2002. U.S.Department of Commerce/National Institute of Standards and Technology.

[115] USTEM. Universitaere Service-Einrichtung fuer Transmissionselektronenmikroskopie. `http://www.ustem.tuwien.ac.at/EN`. [Online; accessed 19-August-2014].

[116] Mathy Vanhoef and Frank Piessens. Practical Verification of WPA-TKIP Vulnerabilities. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 427–436, New York, NY, USA, 2013. ACM.

[117] P.P. Varaiya, F.F. Wu, and J.W. Bialek. Smart operation of smart grid: Risk-limiting dispatch. *Proceedings of the IEEE*, 99(1):40–57, 2011.

[118] Ramakrishnan Venkitaraman and Gopal Gupta. Static program analysis of embedded executable assembly code. In *Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '04, pages 157–166, New York, NY, USA, 2004. ACM.

[119] J. Viega and H. Thompson. The state of embedded-device security (spoiler alert: It's bad). *Security Privacy, IEEE*, 10(5):68–70, 2012.

[120] R. Vigo, E. Yuksel, and C.D.P.K. Ramli. Smart grid security a smart meter-centric perspective. In *Telecommunications Forum (TELFOR), 2012 20th*, pages 127–130, 2012.

[121] Shyh-Yih Wang and Chi-Sung Laih. Efficient key distribution for access control in pay-tv systems. *IEEE Transactions on Multimedia*, 10(3):480–492, 2008.

[122] Dong Wei, Yan Lu, Mohsen Jafari, Paul Skare, and Kenneth Rohde. An integrated security system of protecting smart grid against cyber attacks. In *Innovative Smart Grid Tech.*, pages 1–7, January 2010.

[123] Steve H. Weingart. *Cryptographic Hardware and Embedded Systems — CHES 2000: Second International Workshop Worcester, MA, USA, August 17–18, 2000 Proceedings*, chapter Physical Security Devices for Computer Subsystems: A Survey of Attacks and Defenses, pages 302–317. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[124] Mark Weiser. Program slicing. In *Proceedings of the 5th International Conference on Software Engineering*, ICSE '81, pages 439–449, Piscataway, NJ, USA, 1981. IEEE Press.

[125] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publishing Company, USA, 4th edition, 2010.

[126] Wigle.net Project. `https://wigle.net`. [Online; accessed 13-Nov-2015].

[127] C. Willems, T. Holz, and F. Freiling. Toward automated dynamic malware analysis using cwsandbox. *Security Privacy, IEEE*, 5(2):32–39, 2007.

[128] S. Wong. The evolution of wireless security in 802.11 networks: Wep, wpa and 802.11 standards. *http://www.sans.org/rr/whitepapers/wireless/1109. php (retrieved 2016-03-24)*, 2003.

[129] Xilinx Inc. DS160, v2.0, Spartan-6 Family Overview. `http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf`, 2011. [Online; accessed 22-April-2015].

[130] Ye Yan, Yi Qian, Hamid Sharif, and David Tipper. A survey on cyber security for smart grid communications. *Communications Surveys Tutorials, IEEE*, 14(4):998–1010, 2012.

[131] Bo Yang. Secure Scan: A Design-for-Test Architecture for Crypto Chips. In *in Proc. of 42nd Annual Conference on Design Automation*, pages 135–140. ACM Press, 2005.

[132] Bo Yang, Kaijie Wu, and Ramesh Karri. Scan Based Side Channel Attack on Dedicated Hardware Implementations of Data Encryption Standard. In *in Proc. of the IEEE Int. Test Conf. (ITC), 2004*, pages 339–344, 2004.

[133] Wang Yufei, Zhang Bo, Lin WeiMin, and Zhang Tao. Smart grid information security - a research on standards. In *Advanced Power System Automation and Protection (APAP), 2011 International Conference on*, volume 2, pages 1188–1194, 2011.

[134] Jonas Zaddach, Luca Bruno, Aurelien Francillon, and Davide Balzarotti. Avatar: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares. In *Network and Distributed System Security (NDSS) Symposium*, NDSS 14, February 2014.

[135] ZTEX Gmbh. Products. `http://www.ztex.de`. [Online; accessed 22-April-2015].