Université de Neuchâtel
Faculté des Sciences
Institut d'Informatique

On the Challenges of Energy Efficiency, Scalability and Security for Internet of Things Services

par
Christian Göttel

Thèse soutenue à la Faculté des Sciences le 21.02.2022 pour l'obtention du grade de Docteur ès Sciences

Acceptée sur proposition du jury:

Prof. Pascal Felber, directeur de thèse
Université de Neuchâtel, Suisse

Dr. Valerio Schiavoni, co-directeur de thèse
Université de Neuchâtel, Suisse

Prof. Rüdiger Kapitza
Technische Universität Braunschweig, Allemagne

Prof. Romain Rouvoy
Université de Lille, France

Prof. Christian Cachin
Université de Berne, Suisse

UNIVERSITÉ DE NEUCHÂTEL

Faculté des Sciences
Secrétariat-décanat de Faculté
Rue Emile-Argand 11
2000 Neuchâtel – Suisse
Tél : + 41 (0)32 718 21 00
E-mail : secretariat.sciences@unine.ch

IMPRIMATUR POUR THESE DE DOCTORAT

La Faculté des sciences de l'Université de Neuchâtel autorise l'impression de la présente thèse soutenue par

Monsieur Christian GOETTEL

Titre:

"On the Challenges of Energy Efficiency, Scalability and Security for Internet of Things Services"

sur le rapport des membres du jury composé comme suit:

- Prof. Pascal Felber, directeur de thèse, Université de Neuchâtel, Suisse
- Dr Valerio Schiavoni, co-directeur de thèse, Université de Neuchâtel, Suisse
- Prof. Rüdiger Kapitza, TU Braunschweig, Allemagne
- Prof. Romain Rouvoy, Université de Lille, France
- Prof. Christian Cachin, Université de Berne, Suisse

Neuchâtel, le 22 mars 2022

Le Doyen, Prof. A. Bangerter

To my family and friends

Abstract

The frontier of the Internet is a highly volatile and rapidly evolving area with Internet of Things (IoT), mobile, and smart devices. Every day more and more devices are being deployed and connected to clouds in order to form distributed systems at any scale. Especially large-scale IoT systems make use of public cloud infrastructure for demanding IoT data processing. As a result of this continuous expansion, common network infrastructures and cloud data centers have to shoulder a greater part of expanding Internet traffic. Furthermore, with their growing number of features, these systems offer an increasingly large attack surface to cybercriminals and malicious users. Once deployed, IoT devices are constantly exposed to a wide range of threats. Unauthorized persons can gain physical access to devices or devices can be compromised remotely by software exploits and vulnerabilities over the network.

The increasing availability of hardware-based trusted execution environments, ranging from low-power embedded devices to high-performance servers, could be a solution to these challenges. In particular the lack of trust towards cloud providers has limited the deployment of large-scale distributed systems for manipulating confidential and sensitive data. Public and private companies, as well as organizations or individuals, are taking major risks by exposing their assets and business interactions to the cloud. Under some circumstances, it can even be illegal to disclose certain data or documents to the public, such as medical records. Exploiting trusted execution environments enables dissemination and processing of confidential and sensitive data in the cloud without having to trust the cloud provider. Alternative solutions, such as homomorphic encryption, address the same trust issues but remain in their current state in terms of computational performance and energy consumption behind requirements of current workloads. Most importantly, hardware heterogeneity of trusted execution environments allows selecting platforms based on the energy requirements of software services.

Hardware, firmware, and operating system support is necessary for enabling complex interactions between various components in order to provide fully functional trusted execution environments. Software frameworks abstract the underlying complexity of these interactions and provide user space programming interfaces for large-scale adaptation of trusted execution environments. Understanding how IoT services shielded by trusted execution environments are being affected in terms of energy efficiency, scalability and security remains mostly unclear. This knowledge is particularly important for developing secure and scalable IoT services that run on battery-powered devices as well as to extend the availability and uptime of services.

Many IoT devices rely on battery power or renewable energy sources, such as solar energy through solar panels. Critical supply voltage levels are reached when the battery is nearly empty or if insufficient solar power is provided, allowing for locally or remotely deployed voltages glitches to compromise security guarantees of devices. Under these conditions the integrity and security of IoT credentials, data, services and the system as a whole are at stake. Therefore, not only security, but also energy levels have to be considered when designing such systems, since particularly in IoT, energy and security are strongly intertwined.

Novel approaches need to be considered for IoT services deployed in public and private clouds for determining safe undervolting margins and detecting maliciously undervolted cloud instances.

The current thesis presents a selection of security-relevant IoT services which are enhanced by trusted execution environments in order to provide better security guarantees. These IoT services include design, implementation, deployment, and evaluation of blockchain, key-value and secure storage, as well as network performance systems on low-power ARM-based hardware. Energy efficiency, scalability, and security compromises are quantified in small-scale deployments for the evaluation. The secure IoT services are complemented by two novel energy-related approaches. The first approach is motivating the use of generalized deduplication in IoT network protocols to reduce energy, transmission, as well as storage costs. Undermining the reliability of processors is explored in a second approach where an undervolted cloud infrastructure is considered in an attack. The primary interest of this thesis are low-power IoT devices using ARM TrustZone in combination with the OP-TEE framework as trusted execution environment.

Keywords: blockchain, cloud, distributed system, energy, IoT, scalability, security, TEE

Résumé

La frontière de l'Internet est un domaine extrêmement volatile et évolue rapidement qui est constitué d'appareils *Internet des objets* (IoT), mobiles et intelligents. Chaque jour, de nouveaux appareils sont déployés et connectés aux clouds pour former des systèmes distribués de toute échelle. Les infrastructures des clouds publics sont principalement utilisées par les grands systèmes distribués pour le traitement exigeant des données issue de l'IoT. Ce développement continu a comme conséquence l'augmentation drastique d'une partie du trafic Internet, et doit être gérée par l'infrastructure des réseaux communs ainsi que les centres de données des clouds. De plus, le nombre de fonctionnalités de ces systèmes est croissant, ce qui offre aux cybercriminels des opportunités d'attaques de plus en plus fréquemment. Une fois déployés, les appareils IoT sont constamment exposés à une grande variété de menaces. Ainsi, des intrus pourraient accéder aux appareils potentiellement compromis par des exploits et des failles de sécurité à travers les réseaux.

La disponibilité croissante des environnements d'exécution de confiance dans l'équipement informatique, allant des systèmes embarqués de faible puissance aux superordinateurs, peut être une solution à ces défis. En particulier, un manque de confiance dans les hébergeurs a limité le déploiement des systèmes distribués à grande échelle, lorsqu'ils manipulent des données confidentielles et sensibles. Les entreprises privées et publiques, les organisations et les individus prennent des risques majeurs en exposant leurs actifs et interactions commerciales aux services informatiques hébergés. Dans certains cas, il est illégal de rendre certaines données et documents publique, comme par exemple les dossiers médicales. L'exploitation des environnements d'exécution de confiance permet la diffusion et le traitement des données confidentielles et sensibles dans les clouds, sans pour autant devoir faire confiance aux hébergeurs. Les solutions alternatives, telles que le chiffrement homomorphe, adressent les mêmes problèmes de confiance, mais sont dans inférieurs aux attentes courantes en termes de performance de calcul et d'efficacité énergétique. Pourtant, l'hétérogénéité du matériel informatique actuel et des environnements d'exécution permet aujourd'hui de sélectionner les plate-formes d'après les besoins énergétiques des services logiciels.

Le support du matériel informatique, du micrologiciel et des systèmes d'exploitation est nécessaire pour permettre des interactions complexes entre les divers composants, afin d'offrir des environnements d'exécution de confiance entièrement fonctionnels. Les frameworks logiciels réduisent la complexité sous-jacente de ces interactions et offrent à l'espace utilisateur des interfaces de programmation pour l'adaptation à grande échelle des environnements d'exécution de confiance. Cependant, l'impact en terme d'efficacité énergétique, de passage à l'échelle et de sécurité sur ces services IoT protégés par les environnements d'exécution de confiance demeure inconnu. Hors, cette expertise est particulièrement importante pour développer des services IoT sécurisés et extensibles qui exécutent sur des appareils alimentés par batterie, ainsi que pour étendre la disponibilité de ces appareils et de leurs services.

De nombreux appareils IoT dépendent d'une batterie ou d'une source d'énergie renouvelable, telle que l'énergie solaire, grâce à des panneaux photovoltaïques. La tension électrique atteint des niveaux critiques lorsque la batterie est déchargée, ou lorsque l'énergie solaire récupérée est insuffisante. Dans ces conditions particulières, il est possible d'exploiter des impulsions parasites en local ou à distance afin de compromettre les garanties de sécurité des appareils. Ainsi, l'intégrité et la sécurité des identifiants d'authentification, des données issues de l'IoT, et du système en lui-même peuvent être compromises. Pour cette raison, la sécurité mais aussi les niveaux d'alimentation en énergie doivent être pris en compte lorsque ces systèmes sont conçus, puisqu'ils sont étroitement liés avec le mode de fonctionnement des périphériques IoT. De nouvelles approches sont à prendre en considération pour les services IoT déployés dans les clouds publics et privés, afin de déterminer des marges de tension sûres et de détecter des instances hébergés pour lesquels la tension a été réduit par malveillance.

Cette thèse présente une sélection de services IoT, qui mettent en valeur les environnements d'exécution de confiance pour d'offrir de meilleures garanties de sécurité. Ces services IoT incluent la conception, l'implémentation, le déploiement et l'évaluation des systèmes de blockchain, de stockage clé-valeur et sécurisé ainsi que de performance réseau sur la base des processeurs ARM. Pendant l'évaluation, l'efficacité énergétique, la scalabilité et la sécurité sont quantifiées pour des déploiements à petite échelle. Les services IoT de sécurité sont complétés par deux approches novatrices liés à l'énergie. La première présente le principe de la déduplication généralisée aux appliquée aux protocoles de communication IoT pour réduire les coûts d'énergie, de transmission et de stockage. La seconde traite de l'impact de potentielles attaques par réduction de tension sur la fiabilité des processeurs. Cette thèse se focalise principalement sur le cas des appareils IoT de faible puissance qui utilisent l'environnement d'exécution de confiance ARM TrustZone, en combinaison avec le framework logiciel OP-TEE.

Mots clés : blockchain, énergie, IdO, nuages, scalabilité, sécurité, système réparti, TEE

Acknowledgments

Life can only be understood backwards; but it must be lived forwards.

Søren Kierkegaard

I wish to express my sincere gratitude to everyone who supported me in the accomplishment leading to this thesis. First and foremost I am very grateful to the computer science department of the University of Neuchâtel and the European Union's Horizon 2020 project LEGaTO for the funding of the current thesis. I am in particular thankful to the supervisors of my thesis Valerio Schiavoni and Pascal Felber, who have given me the opportunity to persue my doctoral research studies. Their advice, encouragement and feedback have been most helpful resulting in the publication of several articles. I would also like to thank my colleague Isabelly Rocha with whom I shared the office. We have been working together on the LEGaTO project and have co-authored several publications. I have highly valued our discussions as well as the experience we shared during our work, and that maybe some day I get the chance to challenge her again in a game of chess.

My knowledge on trusted execution environments has greatly benefited from the experience and advice of my colleagues Rafael Pires and Sébastien Vaucher who have been working in depth with this technology. Marcelo Pasin has been a great help while collaborating on the LEGaTO project. He has been coordinating a lot of our joint work and managed most of our contributions to the project. It has been a real pleasure to be traveling with a true polyglot to project meetings. I would also like to thank Dorian Burihabwa, Jämes Ménétrey, Rémi Dulong and Sébastien Vaucher, for all the administration work they have dedicated to the maintenance of the department cluster. I very much appreciated their tolerance and patience when looking at my rack-to-rack overhead cable management I occasionally installed in the cluster for my power consumption measurements.

The current thesis also contains contributions from external researches. Generalized deduplication is the area of expertise of my colleagues Niloofar Yazdani and Lars Nielsen under supervision of Daniel E. Lucani from Aarhus University. I have greatly enjoyed our collaboration and the experience we shared from implementing generalized deduplication into the Hermes protocol. During my thesis I could also assist the master thesis of Christina Müller from University of Bern on implementing the Fabric OP-TEE Chaincode prototype. Marcus Brandenburger from IBM Zürich has been helping out for setting up a Hyperledger Fabric network in our cluster, which allowed us to do the energy and performance evaluations in TZ4Fabric. The experience and knowledge that Osman S. Ünsal from Barcelona Supercomputing Center and Konstantinos Parasyris from Lawrence Livermore National Laboratory have on energy efficiency has been very valuable for implementing the Scrooge Attack and evaluating the reliability of undervolted cloud instances.

Further, I would like to thank Mascha Kurpicz and Patrick Marlier from

University of Neuchâtel as well as Guillaume Fieni from Centre Inria of the University of Lille for implementing and providing useful tools to record power consumption measurements. I have been using the `powerspy.py` script of Mascha and Patrick for most of my power consumption measurements with Raspberry Pis. The Python script of Guillaume has allowed me to measure the power consumption of the servers in our cluster. Moreover, I thank the following people for the numerous moments we shared together: Emilie Auclair, Roberta Barbi, Maria Carpen-Amarie, Andreia Correia, Vero Estrada-Galiñanes, Raluca Halalai, Yaroslav Hayduk, Catherine Ikae, Mirco Kocher, Peter Kropf, Andrei Lapin, Lorenzo Leonini, Umberto Martinez-Peñas, Hugues Mercier, Sukanya Nath, Jacques Savoy, Nils, Schaetti, Peterson Yuhala.

Finally, I thank Prof. Rüdiger Kapitza, Prof. Romain Rouvoy and Prof. Christian Cachin for accepting to be part of the jury and dedicating their time and efforts to the evaluation of my thesis. Their feedback has been very helpful in the finalization of my work and our discussion has shown me many additional directions for continuing this work.

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the LEGaTO Project (legato-project.eu), grant agreement No 780681.

Contents

List of Figures

List of Tables

List of Abbreviations

API Application Programming Interface
ASIC Application-Specific Integrated Circuit
ASID Address Space Identifier
AVS Adaptive Voltage Scaling

BFT Byzantine Fault Tolerance
BIOS Basic Input/Output System

CA Certification Authority
CApp Client Application
CCF Confidential Consortium Framework
CDN Content Deliver Network
CISC Complex Instruction Set Computer
CPU Central Processing Unit
CRC Cyclic Redundancy Check
CRTM Core Root of Trust for Measurement

DAG Directed Acyclic Graph
DDoS Distributed Denial-of-Service
DLT Distributed Ledger Technology
DoS Denial-of-Service
DSA Digital Signature Algorithm
DSL Domain-Specific Language
DVFS Dynamic Voltage and Frequency Scaling

ECC Error Correction Code
EDP Energy-Delay Product
EE Execution Environment
EK Endorsement Key
EL Exception Level
ELF Executable and Linkable Format
eMMC embedded MultiMediaCard
EPC Enclave Page Cache
eRoTc Extended Root of Trust Component
ES Encrypted State
ETR Energy Throughput Ratio
EVM Ethereum Virtual Machine

FOC Fabric OP-TEE Chaincode
FPC Fabric Private Chaincode
FPGA Field-Programmable Gate Array

GD Generalized Deduplication
GPU Graphics Processing Unit

HUK Hardware Unique Key

IaaS Infrastructure as a Service
IoT Internet of Things
IPC Interprocess Communication
IPv4 Internet Protocol version 4
IPv6 Internet Protocol version 6
iRoTc Initial Root of Trust Component
ISA Instruction Set Architecture

JVM Java Virtual Machine

LPDDR Low-Power Double Data Rate

MEE Memory Encryption Engine
METR Microprocessor ETR
MITM Man-in-the-Middle
MOM Message-Oriented Middleware
MSR Model-Specific Register

NDA Non-Disclosure Agreement
NFC Near-Field Communication
NIC Network Interface Controller
NS Non-Secure

OEM Original Equipment Manufacturer
OPP Operating Performance Point
OP-TEE Open Portable Trusted Execution Environment
OS Operating System

PBFT Practical Byzantine Fault Tolerance
PCR Platform Configuration Register
PDU Power Distribution Unit
PITM Person-in-the-Middle
PoE Power over Ethernet

PoS Proof of Stake
PoW Proof of Work
PTA Pseudo Trusted Application

QoS Quality of Service

REE Rich Execution Environment
REST Representational State Transfer
RFID Radio-Frequency Identification
RISC Reduced Instruction Set Computer
RMP Reverse Map Table
RMS Root Mean Square
ROM Read-Only Memory
RoT Root of Trust
RPC Remote Procedure Call
RSA Rivest-Shamir-Adleman
RTM Root of Trust for Measurements
RTR Root of Trust for Reporting
RTS Root of Trust for Storage

SDK Software Development Kit
SDRAM Synchronous Dynamic Random Access Memory
SEV Secure Encrypted Virtualization
SGX Software Guard Extensions
SLA Service-Level Agreement
SMC Secure Monitor Call
SME Secure Memory Encryption
SMP Symmetric Multiprocessing
SMR State Machine Replication
SNP Secure Nested Paging
SoC System on a Chip
SP Secure Processor

TA Trusted Application
TC Trusted Computing
TCB Trusted Computing Base
TCG Trusted Computing Group
TEE Trusted Execution Environment
TPM Trusted Platform Module

URI Uniform Resource Identifier
UTXO Unspent Transaction Output

VM Virtual Machine

1 Introduction

1.1 Context

Computers have become ubiquitous in our every day life. Fictitious gadgets in old movies have now become reality. The technological advance in nano-electronics have made it possible to continuously shrink microprocessor[1] sizes while also improving their computational performance and energy efficiency. For decades processors have steadily become faster and more energy efficient with every generation, from which the software market has benefited the most. However, the free ride has come to an end as the physical limits of processors are being reached. Consequently processor designs are shifting away from general purpose architectures [137] towards domain-specific and heterogeneous architectures in order to provide dedicated performance improvements and further enhance energy efficiency.

As a result of this architectural shift, an increasing number of processors is nowadays shipped with additional coprocessors. These coprocessors provide specific functionalities that on the one hand greatly improve energy efficiency, latency and throughput of dedicated software and on the other hand are inflexible because integrated circuits cannot be changed after manufacturing. It is because of coprocessors that many devices are nowadays able to connect to the Internet or can be linked directly with other devices. Other coprocessors provide cryptographic techniques and hardware-based security solutions, in order to protect data from preying eyes and shield it from malicious attacks or compromised systems.

These hardware-based security solutions are commonly known as Trusted Execution Environments (TEEs) and vary in the number of security features and the extend of security guarantees they provide. TEEs require software support from firmware, Operating System (OS) [287], and user space to fully exploit their features. All types of platforms, including embedded and mobile, desktop as well as server platforms, are integrating TEEs. Each TEE is targeting different subsets of platforms such that distributed systems [138] need to provide support for multiple TEEs if they are using various platforms and relying on hardware-based security. Unfortunately, due to architectural and design differences in hardware implementations of TEEs, developers can only make use of frameworks with partial support to conveniently port applications to different TEEs[2] despite the existence of Application Programming Interfaces (APIs) and

[1] From here on abbreviated to *processor*.

[2] The Open Enclave SDK [89, 90], initially open sourced by Microsoft, aims to establish such a framework but, at the time of writing, provides support only for Intel SGX and a preview for OP-TEE on ARM TrustZone[3]. Google Asylo [253] is another framework which is

standards. Therefore, applications running in TEEs, that do not strictly adhere to standards and make use of extensions, are vendor specific and remain incompatible as well as non-portable to other TEE vendors.

Trust issues that exist between mutually distrusting parties can be resolved through the use of TEEs. Blockchains are an interesting example for distributed services bringing together all kinds of people which share common interests but do not necessarily trust each other. Nevertheless, people already transfer cryptocurriencies, trade assets, or track supply chains by using blockchains on public or private networks. These transactions can, for example, be written down and controlled by smart contracts [286] that are stored on the blockchain. Distributed Ledger Technology (DLT), a generalization of blockchains, is attributed a key role in the realization of the industry 4.0. This is one of the reasons many cloud providers, companies, and institutions are heavily innovating, in particular in smart contracts. Deploying smart contracts on various platforms and integrating TEEs into blockchain systems are in the current focus of research. Energy efficiency is particularly important for smart contract solutions deployed on embedded, mobile and Internet of Things (IoT) devices. Appropriate hardware and software support are necessary in order to realize IoT blockchain systems that provide both security and energy efficiency.

Public clouds [303] are the platform of choice for large-scale IoT systems and microservices [157]. The Infrastructure as a Service (IaaS) provided by clouds offers users the flexibility to rent resources in the form of containers, Virtual Machines (VMs), and bare-metal instances to deploy their own services. Heterogeneous resources in the form of specialized hardware allow users to select instances based on the needs and requirements of their services. At any time can these resources be scaled either horizontally or vertically. While much less flexible than cloud computing, edge computing [237] offers similar functionalities on low-power and resource limited devices with the advantage of being located closer to the data source. Innovative network traffic and storage solutions are needed to take on the many challenges introduced by the continuous deployment of IoT devices and services for the industry 4.0.

Power management mechanisms, such as voltage and frequency scaling, are being used for enabling low power idle states, prevent degrading or damaging processors from excessive heat dissipation, and saving battery power. While these energy savings appear to be marginal on single devices, they matter specifically for large-scale deployments such as in data centers or IoT. Even low-power processors, primarily designed for battery-powered embedded and mobile devices, have found their way into data centers and can be rented as instances. The design of the newly introduced Neoverse [3, 32] server-grade ARM microarchitecture has also been driven by consumer needs and market demand. Early ARM server platforms have been based on modified application-grade or custom-designed ARM microarchitectures. Cloud providers even experimented with offering collocation of low-power devices to free up resources on servers and to deploy less demanding workloads on these devices. The French cloud provider Scaleway went as far as to design their own custom credit-card sized ARM instances [195]. Diversity of cloud offerings is becoming larger with increasing heterogeneity of cloud platform hardware. Cloud instances can now be rented based on the individual requirements of services, which allows for

only providing support for Intel SGX and has not seen any significant updates recently.

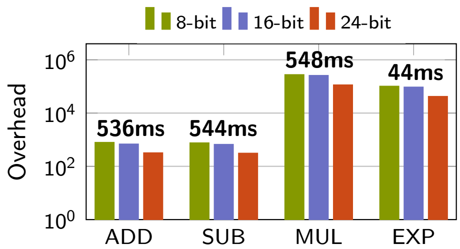


Figure 1.1: Performance overhead of fully homomorphic encryption operations for three data sizes. Numbers above bars indicate average latency of operations.

energy-efficient deployments and cost savings by both the cloud provider and cloud user.

1.2 Challenges

Several challenges related to the design of energy-efficient, scalable and secure IoT services are addressed by this thesis. The importance of these challenges is increasing as a growing number of heterogeneous devices is being deployed at the edge of the Internet. Network conditions for IoT services are continuously changing and require scalable solutions that can quickly adjust in order to improve energy efficiency and reduce latency on devices processing end user data. In particular the use of TEEs for preserving user data privacy by IoT services is essential for energy-efficient implementations. The following paragraphs discuss the challenges that are being addressed by this thesis.

Privacy Concerns Regarding Confidential IoT Data. IoT and smart devices have many functionalities at hand which allow to detect and track things and persons in their proximity. The data smart devices collect on persons over sensors contains sensitive information which can also be shared with public clouds for processing and storage. The question arises whether this sensor data, that IoT and smart devices access, collect, process and store, is sufficiently secured. Would malicious attackers be able to make use of exploits and vulnerabilities in IoT and smart devices to eavesdrop on objects or even persons? As much as this revolutionary technology bears great potential for innovative and novel use cases, it can also be misused in an Orwellian way.

Privacy concerns are limiting the deployment of IoT systems and microservices on public clouds, and instead IoT systems and microservices are being deployed at the edge. Cryptographic solutions can provide the necessary protection for confidential and sensitive data without having to establish trusted relationship with infrastructure providers. Software-based solutions such as partially homomorphic encryption [229, 230, 285] or asymmetric scalar product-preserving encryption [76] have been proposed to address these privacy concerns. The energy and performance impact of these software-based solutions is so severe that realistic workloads become impracticable. For this reason, it is currently expensive to use homomorphic encryption at the edge or even in the cloud to secure confidential and sensitive data. Figure 1.1 demonstrates that the performance of operations using fully homomorphic encryption is orders of magnitude behind performance requirements. By contrast, TEEs are much less computationally and energy demanding than software-based solutions. Independent of hardware-based or software-based approaches, services have to be adapted in order to be able to shield confidential and sensitive data, which requires additional engineering efforts impacting energy consumption and performance. The impact that TEEs, and in particular TrustZone with Open Portable Trusted Execution Environment (OP-TEE) [187], have on IoT services remains unknown and has so far not been addressed in the literature.

Privacy Concerns Regarding Smart Contracts. Novel business interactions are being enabled by deploying smart contracts on embedded, mobile or IoT devices and especially of interest to the industry 4.0. These devices are resource limited and constantly exposed to threats, which makes executing smart contracts on them particularly challenging. Furthermore, the original design of smart contracts publicly discloses the contents of contracts, data and transactions and is not providing confidentiality. TEEs provide the necessary means to keep smart contracts as well as assets and business interactions confidential. In order to prevent TEEs from being rolled back [52], forks in the blockchain have to be avoided by relying on final consensus protocols in private blockchains.

Private blockchains are favorable for embedded, mobile and IoT devices because more energy efficient consensus protocols can be used. Public blockchains, such as Bitcoin [231] and Ethereum [59], are using Proof of Work (PoW) consensus which requires solving cryptographic puzzles. These cryptographic puzzles are computationally expensive and consume a lot of energy for solving. The greed for profit by mining new blocks and be rewarded with fresh tokens has raised the total energy consumption of public blockchain networks and is now on par with small to middle-sized European countries. Designing secure IoT services around private blockchain solutions for providing confidential smart contract execution through the use of TEEs remains challenging.

Managing Future IoT Network Traffic. The amount of generated IoT data is increasing as the number of deployed IoT devices is continuously rising. IoT devices have very limited resources and are unable to store the majority of generated IoT data for longer time periods. The only solution to process, analyze, and store these large volumes of IoT data is the offload to external resources such as available in the cloud. The consequences are additional IoT traffic and load on the Internet. The issue with current approaches is that many network protocols are considering only individual IoT devices and the cloud instance in point-to-point connections and not the system as a whole.

Compressing IoT data is resulting in the reduction of IoT traffic and saving network bandwidth. However, efficient compression algorithms are computationally demanding and greedy for memory, which is standing in contrast to the capabilities of IoT devices. Furthermore, IoT data consists of small data chunks that do not provide sufficient information required for efficient compression. For this reason, the choices for suitable compression algorithms or parameters are restricted by the IoT device capabilities and IoT data chunk sizes.

Energy-Efficient Operation of Processors. Undervolting is an inexpensive solution, where the supply voltage is set below nominal voltage, in order to reduce the energy consumption of devices and increase battery lifetime. Process variation during processor manufacturing as well as lack of resources to individually adjust appropriate voltage levels for each and every processor are reasons for conservative voltage margins. However, undervolting comes at the cost of reliability by introducing soft and hard errors due to timing violations which undermine integrity and security of systems. Side-channel attacks can exploit undervolting by enforcing particular program control flows with voltage glitches, break kernel address-space layout randomization [192] or extract encryption keys from TEEs [228] in order to mitigate security guarantees and compromise systems.

Temperature has always been a major issue since the invention of processors which results from frequency and voltage settings during processor operation. Although supply voltages could be steadily reduced by shrinking transistor sizes over the last decades, the overall energy density on processors has increased. The problems are that neither transistors can be made arbitrarily small, nor supply voltages can be set below threshold voltages. These physical limits are already interfering with processor designs and are leading to new domain-specific architectures with dedicated accelerators to maintain energy efficiency and performance improvements.

1.3 Contributions

The contribution of this thesis are addressing various combinations of energy-efficient, scalable and secure designs for IoT services. TEEs provide security guarantees that are being integrated in an increasing number of processors. Understanding the impact that TEEs have is crucial in the design of secure IoT services. Ignoring these limitations can severely reduce the energy efficiency and scalability of secure IoT services. Energy efficiency is especially important to battery-powered embedded and mobile devices. IoT services are directly influencing the energy consumption of devices which in turn is affecting the availability and uptime of services. Protocols are taking a key role in the design of energy-efficient and scalable services for IoT. Undervolting can further contribute to the energy efficiency of IoT services but needs to be applied carefully in order to not undermine the reliability of the services. The following complementary contributions are presented in this thesis.

Protecting IoT Services With Trusted Execution Environments. Shielding IoT services using TEEs is impacting IoT devices and systems as a whole. In particular the question of *how exactly TrustZone with OP-TEE are affecting the energy efficiency, scalability and security of IoT services* is being addressed. Compromises in terms of energy efficiency, scalability, and security for shielding selected IoT services have been quantified. These secure IoT services include secure storage [123], network performance [124] and blockchain [227] systems for which prototypes are implemented and evaluated in small-scale deployments. It is shown how IoT services can benefit from TEEs and how security-based design decisions can negatively impact energy consumption and scalability of systems by introducing bottlenecks. These bottlenecks, such as the message passing overhead in the TrustZone, can limit the scope of deployable services. The `iperfTZ` network performance tool has been created specifically to assist in identifying these bottlenecks early in the development cycle of network-enabled services. By adhering to the limitations of TEEs in general, IoT services gain in terms of security with reasonable losses in energy efficiency and scalability. Chapter 3 explains in detail what these limitations of TEEs are and how they can be avoided for specific IoT services.

Confidential Smart Contracts. The increasing number of IoT devices with TEEs is an enabling factor for adding confidentiality guarantees to smart contracts and thriving novel business relations for the industry 4.0, provided national and international laws make it possible. Assets and business relations are no longer publicly exposed when shielding the execution of smart contracts with TEEs. Smart contracts are executing the agreed terms by parties according to transaction protocols. Computational and network resources are required within the TEE in order to verify contractual conditions and transmit transactions. The research question addressed by this contribution is *how to develop energy-efficient, scalable and secure IoT services.* Necessary knowledge needed for designing energy-efficient, scalable and secure IoT services making use of TEEs is being provided by the previous contribution. This knowledge is applied for developing an IoT prototype that isolates the smart contract execution within the TEE.

Fabric OP-TEE Chaincode (FOC) [227] is a prototype based on Hyperledger Fabric[4] [25] which separates the smart contract execution from the system and shields it in TEEs of IoT devices. Confidential smart contracts leverage TEEs for protecting the contract, data and transactions from threats. This allows, for example, TEE-enabled sensor nodes to shield the execution of smart contracts and simultaneously provide secure access to sensor values for validating conditions in the contract. Supply chains can make use of confidential smart contracts to monitor transport conditions of goods that are sensitive to humidity or temperature such as chemicals, drugs or vaccines. The development of a secure IoT service and its evaluation for energy efficiency and scalability are exemplified with the FOC prototype in chapter 4.

Resource-Efficient Compression of IoT Data With Generalized Deduplication. Deduplication [283] and network coding [312] are complementary techniques to compression which are not subjected to limitations of IoT devices. In particular network coding is worthwhile considering given the unreliable nature of IoT device network connectivity. Alternatively, Error Correction Codes (ECCs) are less demanding and ideal for IoT use cases compared to compression algorithms. Chapter 5 is answering the question on *how to design an IoT protocol that integrates energy-efficient techniques in a scalable way.*

Hermes [125] is an energy-efficient network protocol which scales across multiple devices and exploits Generalized Deduplication (GD) [298] in order to reduce IoT traffic and cloud storage costs. GD is a novel compression technique that, unlike classical deduplication, can map data to identical and similar data by leveraging ECCs. By expanding the data volume, GD capitalizes on additional data correlations to further improve compression ratios. GD has the

[4]From here on referred to as Fabric.

potential to reduce transmission and storage costs of IoT data while providing better energy efficiency than common compression approaches and embracing edge computing. Hermes deviates from state of the art network protocols using point-to-point connections or broker systems for relaying IoT traffic. The network protocol instead establishes a hierarchical hop-by-hop network to increase the probability of matching IoT data to identical or similar IoT data. Hop-by-hop approaches can evaluate partial IoT traffic at different intermediate nodes and include multiple IoT devices in the connection.

Remote Identification of Undervolted Cloud Instances Through Service-Level Agreement Violation. Improving the energy efficiency of computing devices is becoming more challenging with current approaches, such as Dynamic Voltage and Frequency Scaling (DVFS), as their energy savings are diminishing with newer processor generations. Undervolting is leveraging conservative voltage margins to improve energy efficiency by reducing supply voltages to safe minimum voltages. Operating processors at reduced voltage margins, however, lowers the reliability of executed instructions. This is because critical paths are increasing as supply voltages are decreased, which can lead to soft or hard errors during instruction execution. The research question this contribution is addressing *analyzes the consequences of undervolted IoT service infrastructure*.

Clouds and data centers have a particular interest to reduce energy consumption because it also saves energy and costs for cooling. Maliciously undervolted instances pose a threat to cloud and IoT services because it is hard to detect errors. The *Scrooge Attack* [131] is considering such a maliciously undervolted cloud infrastructure and is evaluating the feasibility, stealthiness and energy savings of the attack. Chapter 6 is explains in a first part how cloud providers can run an automated temperature-based guardband analysis setup to determine an undervolting level for their could servers. The second part proposes a detection method that is inspired by the principles of micro and power viruses. The detection method exploits the undermined reliability of services deployed on these undervolted instances to non-selectively inject faults and ultimately crash these instances. Because of the non-selective fault injection it is not possible to attribute the cause of crashes to the cloud user and crashes are therefore covered by the Service-Level Agreement (SLA).

1.4 Organization

The thesis is structured into the following chapters. Chapter 2 summarizes the fundamental concepts and technologies and gives an overview of the scope of this work. Prototypes for secure IoT services based on secure storage, network performance and blockchain are implemented and evaluated in chapters 3 to 4 using TrustZone with OP-TEE as TEE. Chapter 3 is in particular identifying the limitations of the TEE and how they impact energy consumption and performance of shielded services. These findings are then applied in chapter 4 for developing a secure IoT service for smart contracts. The implementation of GD as a compression technique into an energy-efficient and scalable IoT protocol is outlined in chapter 5. Chapter 6 describes the detection method for a novel attack scenario of undervolted cloud instances and discusses consequences of undervolting in general. Finally, the thesis concludes with chapter 7 and gives an outlook on TEEs in IoT for developing secure IoT services with respect to energy efficiency and scalability. Further contributions that are not included in this work are listed in section 7.4.4 of the appendices.

2
Background

2.1 Introduction

This thesis is describing energy efficiency, scalability and security challenges developers are facing when implementing distributed services on IoT devices deployed at the edge. ARM-based devices are utilized in this thesis as IoT devices and their TEE feature is used in order to provide security guarantees. This chapter summarizes fundamental concepts and technologies that are used in the following chapters to design, implement, deploy, and evaluate various IoT services. The chapter first introduces the context of IoT before detailing the challenges of scalability, energy efficiency and security. Monitoring devices, TEEs and the ARM architecture are presented as technologies and tools for dealing with the previously mentioned challenges. The chapter concludes by outlining the specifications of the equipment used for the evaluations.

2.2 Internet of Things

IoT services are next to their challenges on energy efficiency, scalability and security a main object of research in this thesis. Many elements are involved in the IoT, ranging from IoT devices all the way to clouds. In particular the communication between IoT devices and clouds includes complex hierarchies of elements, that are essential to the design of IoT services. It is for this reason that each element within the IoT ecosystem is presented in more detail in this section. The following chapters are each considering different aspects of IoT services as well as the challenges that are related to energy efficiency, scalability and security.

2.2.1 The Internet and Smart Devices

In the late 1960s to the early 1970s, in a time before the *Internet* [64] was broadly established, the first computers were expensive mainframes and generally companies and public institutions were able to acquire. Service requests had to be done in person or were offered over postal mail. More convenient approaches were needed for requests in order to provide shorter service times and better accessibility. The first regional computer networks such as the *ARPANET* [266, 184], *CYCLADES* [254, 255], *MERIT* [36], and *NPL* [93, 272] were assembled and offered universities, research and military institutions remote connections over terminals to their computers in order to request services and access resources available in those networks. Over time, as hardware, software [43], and standards [64, 15] evolved and matured [10], these networks together with newly emerging networks could be interconnected to form by the late 1980s an intercontinental network known as the Internet.

The Internet is a protocol suite that operates on top of different underlying networks using media such as copper, fiber optics, radio waves and satellites for data transmission. In particular human-to-human and human-to-machine interactions shaped the early days of the Internet, where emails were exchanged, files downloaded, or web pages accessed [284]. In the beginning the Internet was mainly a network among computers, whereas other things were not considered and remained disconnected from it. By embedding computers into things such as cars, fridges, LED lamps, smart bulbs and thermostats, they were made "smart" and have become able to connect to the Internet. Suddenly, the boundaries of the Internet are beginning to blur as more and more smart devices connect to and extend the Internet, while the Internet itself moves from the virtual world into our physical world. The technologies enabling this wide spread of computers are diverse, among which the ITU considered the following four technologies to be the most important: Radio-Frequency Identification (RFID), sensors, smart technologies, and nanotechnology [284].

2.2.2 Cloud

The cloud emerged from the grid [108] as a platform offering computing power independent of the location of users. Grid computing had demonstrated that it was no longer necessary to acquire, host and maintain large numbers of servers in-house [303] and that computing resources could be requested on demand. Cloud computing is a paradigm related to the grid computing paradigm that provides centralized computing resources in data centers. The cloud platform allows to dynamically configure and provision computing resources. These computing resources can be either containers, physical or virtual machines with optional resources for storage, network and security needs. The cloud is a pool of virtualized resources [49] which enables fast deployment of services that can be scaled horizontally or vertically and be made redundant. Deployed services are managed as single entities [49] and controlled through web interfaces. Computing resources consumed by users are monitored in real time which allows to make adjustments and reconfigure services in order to reduce costs. The availability, quality and reliability of services provided by clouds are written down in SLAs [249]. SLAs are mutual agreements between cloud users and services offered by cloud providers. These agreements are formulated in such a way that cloud providers have as little liability as possible.

2.2.3 Edge

The edge describes devices at the frontier of the Internet located close to end-users. Such devices include routers, switches and servers with computing and storage facilities. Edge computing is a paradigm which designates the edge as location where data is being processed. Edge devices have the advantage that they are geographically localized and grant low latency to end-users accessing Internet services. Processing data at the edge in the proximity to where it is generated is more efficient than offloading the entire computation to the cloud. Early adopters of edge computing were Content Deliver Networks (CDNs) for caching static content, improving performance of web pages and mitigating

Denial-of-Service (DoS) attacks against web servers. CDNs operate globally distributed systems of servers which spread across thousands of sites worldwide. Autonomous driving cars, gaming, smart cities and smart grids are novel IoT related use cases for which computing resources and low latency are critical factors. Energy, network bandwidth, privacy concerns and time savings [281] are primary reasons for using edge computing over cloud computing solutions. Similar to cloud computing, edge computing is making use of SLAs and Quality of Service (QoS) parameters for services deployed at the edge.

2.2.4 Enabling Technologies

Almost everywhere on the planet and at anytime communication technologies such as 5G, Bluetooth, LoRa, LTE, and Wi-Fi allow the formation of mobile networks. By connecting computers and things to the Internet they become discoverable and can be addressed directly for remote interactions. However, in order to be addressable there is a need for addressing systems that can uniquely identify things. The Internet Protocol version 4 (IPv4) [15] and Internet Protocol version 6 (IPv6) are 32-bit, respectively 128-bit, addressing systems that assigns IP addresses to things. Things disconnected from the Internet can be uniquely identified by Near-Field Communication (NFC) [101] or RFID tags as well as by identification encoded in matrix barcodes. In this case, these things are passive and provide resources to other devices in proximity that need to be equipped with appropriate readers and receivers such as available on smartphones. Geographical positioning and indoor positioning make it possible for things to become aware of their location. Actuators and sensors permit things to interact with their environment, collect information on their surroundings and react to certain events. All these embedded computers are inexpensive, equipped with low-power processors, feature human or machine interfaces and, most importantly, have a network stack to link them up with other things. Only by forming a network and connecting these embedded computers with each other is it possible to realize the real potential of IoT technology. The nature of such networks can be either of local, regional, or global scale as well as interconnected or disconnected character. Independent of their purpose, all things have in common that they are access points to the Internet [202]. Humans and machines interact via embedded computers with things addressable in networks for information retrieval or manipulation of other things. This integration of people with processes and technology to manipulate, monitor and query the state of things equipped with a computer is what in 1985 Peter Lewis called the IoT [181]. Through IoT computers have become ubiquitous. [304]

2.2.5 IoT Devices

The scope of IoT is much larger than the Internet and scalability becomes a primarily challenge not only for network infrastructures, but also in terms of energy, storage and software services [202], even if the communication among IoT devices is often local. It is expected that to some extend IoT devices have an autonomous behavior, or even an artificial intelligence, that allows them to adapt their configuration to the changing environment and establish connections spontaneously [202]. This form of interoperability is only possible, if the IoT devices adhere to standards as well as common practices [202], and are

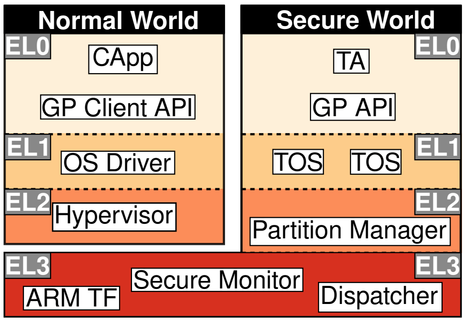


Figure 2.1: Exception levels in the ARMv8-A architecture with the normal and secure world of TrustZone.

fault tolerant by replicating data and services across the network. Discovery of and interaction with IoT devices benefit from the standardization process and enable automatic identification of IoT devices [202] by simplifying communication. The complexity of software binding together IoT devices and taming the generated data volumes is ever increasing. As a result, security issues and privacy concerns are omnipresent in IoT and need to be integrated right from the beginning into the design of IoT devices and IoT services. Energy is at last the limiting factor restricting the operations IoT devices can perform and the types of communication that can be used for data dissemination. Many IoT devices are battery powered and have only a limited amount of available energy. Novel battery technologies and new ways to harvest energy from the environment are necessary [202] to raise this limitation and expand the capabilities of IoT.

2.2.6 The ARM Architecture

ARM processors are based on the Reduced Instruction Set Computer (RISC) architecture and are taking advantage of instruction level parallelism as well as caches for performance. The simple design of instructions allows implementing the architecture using fewer transistors. Area and power savings resulting from lower transistor counts are particularly important for embedded and mobile devices that are battery powered. These are primary reasons why ARM processors are dominating these market segments. Arm Ltd. does not manufacture processors, instead the business model is selling licenses of its semiconductor intellectual property cores to Original Equipment Manufacturers (OEMs). These intellectual property cores include Central Processing Units (CPUs), Graphics Processing Units (GPUs), interconnect fabric, system and peripheral controllers, security, debug and many more. ARMv8 is the current architecture version for application, real-time and microcontroller profile processors.

The ARM application profile architecture is divided into four Exception Levels (ELs) as shown in Figure 2.1. Each EL has different privileges where user space has the lowest privilege (*i.e.*, EL0), kernel space has higher privilege (*i.e.*, EL1) and hypervisor has superior privilege (*i.e.*, EL2). If TrustZone is supported, the system monitor has the highest privilege (*i.e.*, EL3) and all lower ELs are mirrored in the secure world. ARM extended the microarchitecture in ARMv8.4 by an optional secure partition manager [31, 33, 196], that is not being used in previous versions, also because ARM design licensees could not agree on common dispatching mechanisms and software stacks for their firmware, either for intellectual property or other reasons. ELs are changed by either *taking* an exception (*i.e.*, increasing the EL) or *returning from* an exception (*i.e.*, decreasing the EL). Supervisor call (`SVC`), hypervisor call (`HVC`) and secure monitor call (`SMC`) are instructions for taking an exception. The `ERET` instruction returns from an exception after it has been handled at the higher EL.

ELs can either operate in Aarch32 or Aarch64 mode, provided that the higher EL is also operating in Aarch64 mode. The virtual address space has either access to 32-bit or 64-bit address ranges depending on the selected mode for an EL. The L1 cache is part of the CPU intellectual property core and divided into code and data caches. The L2 cache is unified and a separate intellectual property core that is shared by all CPU cores. Cache coherency used by ARM System on a Chips (SoCs) is described by the AMBA ACE protocol [35]. Memory writes are ordered according to the *other-multi-copy atomic* model. Changes written to memory by an observer are perceived coherently by all other observers accessing the memory. It is possible that an observer sees its own writes before they are being made visible to the other observers.

2.3 Scalability

One of the three challenges of IoT services is scalability. Scalability has and is still frequently used as a marketing term for systems, without exactly specifying which of the components in systems are scalable or by how much they can be scaled. The main issue with scalability is that there is no commonly agreed definition for it. This section provides a clear specification on how scalability is defined in the context of this thesis.and what units are being used in the following chapters for quantifying the scalability of IoT services.

2.3.1 Quantifying Scalability

Scalability is a property which describes how systems adapt to changes. The longstanding problem of this property is that there is no common agreement or generally accepted definition for it [142]. There have been several attempts to define scalability, for example, by comparison of parallel with sequential systems [99]. It would be convenient to define scalability with speedup [22] in this case. However, there are several issues when using speedup as a metric to express scalability. Many parallel algorithms have inherent sequential sections that cannot be parallelized, making large systems in particular inefficient. Large systems are also inefficient when problem sizes of benchmarks are held constant and the number of processors exceeds the problem size [109]. Problem sizes could be adjusted to large systems by specifying the factor by which the problem sizes were increased to compare with smaller systems. Alternative proposals argue that execution time should be kept constant [130] instead of keeping problem sizes constant. Both fixed-size and fixed-time speedup are well established metrics in the domain of parallel computing.

2.3.2 Horizontal and Vertical Scalability

Horizontal and vertical scalability are describing ways in which resources are added to or removed from systems. The ability to increase or decrease the number of nodes in a system is specified by horizontal scalability. Horizontally scaled resources execute in parallel and provide the flexibility to adjust to system load. Scaling horizontally under high system load to increase the number of nodes reduces the latency of processed requests.

Vertical scalability is characterizing resources which are local to a device such as computing power, memory, storage or network bandwidth. Adding or removing resources locally on devices is typically done using virtualization technologies, such as VMs or containers, where physical resources are logically abstracted and provided in form of virtual resources. Cloud and edge providers use vertical scaling in order to effortlessly adjust instances to the need of users. Migrating and collocating virtual instances are important techniques for making efficient use of physical resources and provisioning of devices to save energy. Embedded, mobile and IoT devices rarely make use of vertical scalability due to the lack of virtualization technology on these devices.

2.3.3 Scalability of IoT Networks

IoT devices are inexpensive and can be deployed in large numbers for providing exceptional horizontal scalability. The number of globally deployed IoT devices is expected to grow by a factor of $2.4\times$ in five years and reach 14.6 billion devices by 2022 [78]. In order to cope with this increase of IoT devices networks, edges and clouds need to be scalable. 5G is the latest deployed technological standard for broadband cellular networks that provides the necessary infrastructure for enabling more reliable coverage and connectivity to larger numbers of wireless devices. Reliable IoT services have to take into account that network coverage is not always given and that connections for transmitting data can be sporadic. For example changing weather conditions or insufficient battery power are factors which influence the connectivity of wireless sensor networks. Network protocols need to be resilient to connection losses and scalable for ingesting fluctuating IoT traffic at the edge. Edge devices are heterogeneous [48], which makes scalability particularly challenging. Compression algorithms can alleviate scalability issues but need to be fast and have good compression ratio in order to save network bandwidth. Message brokers that are being used by some network protocols for routing messages at the edge have to scale with the network traffic. Clouds with their sheer infinite resources appear to have limitless scalability. Making good use of these resources requires IoT services to be designed as large-scale distributed systems with the flexibility to adapt to changing or irregular conditions.

2.3.4 Evaluating Scalability

Within the context of this thesis scalability is defined by the latency as a function of throughput. Throughput is a system variable that directly depends on the problem size. Systems claim more and more resources as throughput increases with problem size. Resources can be anything from processor cycles, to memory, storage or network bandwidth. Latency determines the time needed by systems to execute operations. As long as systems have enough resources to overcome an increasing problem size, latency will not be affected. Latency will increase as soon as the first resource reaches a bottleneck because of resource exhaustion. This point is referred to as saturation point as systems have reached their peak throughput and are fully saturated. Increasing throughput beyond saturation point results in further increasing latency. Systems are scalable until the saturation point is reached, which marks the maximum scalability achievable by systems. Scalability is expressed in this work in terms of throughput and latency performance metrics also because of their close relation to energy efficiency as explained in the following section.

2.4 Energy Efficiency

Energy efficiency is part of the challenges for the design, implementation and evaluation of IoT services. The utility of energy efficiency in IoT is twofold. On the one hand energy-efficient processor designs are necessary for managing heat dissipation due to the increasing energy density of on processor dies. On the other hand software is making use of energy-efficient mechanisms in order to intelligently distribute power to IoT device components and extend the uptime of battery-powered IoT devices. In the following chapters, energy efficiency is used as a metric for comparing and evaluating IoT services. Energy management mechanisms, such as DVFS or undervolting, are strongly influencing the energy efficiency of IoT services and applications in general. In the following chapters it is demonstrated how energy efficiency of IoT services can be affected through various means.

2.4.1 Energy Efficiency and Dennard Scaling

Energy efficiency, in the context of computer science, is a ratio which quantifies the number of operations executed by a system for a given amount of energy at its disposal. One of the first scaling laws to accurately predict processor parameters that compose energy efficiency, such as delay, power dissipation, and power density, was Dennard scaling [94]. While the industry was keeping pace with Moore's law [220] to double the number of components in processors about every two years, Dennard scaling indicated that every one and a half year energy efficiency doubled at an even faster rate. Later on this observation was also confirmed by Koomey's law [171]. Another driving force for energy-efficient processors was the advent of laptops [165] in the late 1980s. It has become obvious that the battery industry was not innovating as fast as the semiconductor industry. For this reason additional efforts had to be made to further improve energy efficiency with low-power processor designs and extend battery life. Not only hardware, but also software was made more energy-efficient. New mechanisms were introduced by OSs such as dimming and turning off the screen or reducing clock frequency in order to adapt system performance and reduce energy consumption whenever possible.

After 30 years [47] the Dennard scaling came to an end in the mid 2000s when static power consumption became more important than dynamic power consumption for the total processor power consumption. Dennard scaling initially assumed that threshold voltages would scale with supply voltages, which was no longer the case when supply voltages had reached their limits. Supply voltages are limited by the sub-threshold leakage of transistors and the power consumption of processors [47]. Sub-threshold leakage increases as transistors are scaled down which contributes to the static power consumption. These factors were not taken into account when the Dennard scaling was published. A transition from single-core to multi-core processors resulted in order to continue scaling performance with the core count as transistor counts increase. However, the supply voltage scaling [105], which slowed down with the end of Dennard scaling, is also limiting the core count. Multi-core processors are curbed by the power consumption and frequency has to be traded off for core count. Recent processor designs propose heterogeneous cores [129, 301, 87] with performance and energy-efficient cores or accelerated processing units for machine learning.

2.4.2 Energy Efficiency Units

Several energy efficiency units had been proposed [58, 65, 147] in the early 1990s. The most commonly used energy efficiency unit is *energy per operation*. It describes the amount of energy that was spend for executing a single operation. Lower values indicate that operations are more energy-efficient and *vice versa*. Operations are not specifically defined and can be expressed as instructions or programs [58]. This energy efficiency unit can also be expressed as *power per throughput* which directly link it to the throughput used in the definition of scalability given in section 2.3. When comparing two systems to determine the more energy-efficient one, it is necessary to fix the throughput with this unit.

$$\frac{\text{Energy}}{\text{Operation}} = \frac{\text{Power} \cdot \text{s}}{\text{Operations}} = \frac{\text{Power}}{\frac{\text{Operations}}{\text{s}}} = \frac{\text{Power}}{\text{Throughput}}$$

Throughput scales linearly with frequency if the supply voltage is kept constant. It should be noticed from section 2.4.3 that if supply voltage is not kept constant, delay is increased which reduces the number of operations that can be executed per unit of time. If the frequency is inverse to the delay, then throughput is proportional to the *amount of concurrency per clock cycle divided by the delay.* [58] This means that throughput captures the parallelism within an integrated circuit. Supercomputers are constantly running calculations and have a need for low power and high performance in order to solve calculation by spending as little energy as possible. Energy Throughput Ratio (ETR) [58] takes into account this characteristic when expressing energy efficiency for systems operating at maximum throughput. As with the previous energy efficiency unit, lower ETR indicates a more energy-efficient system.

$$ETR = \frac{\frac{\text{Energy}_{\text{max}}}{\text{Operation}_{\text{max}}}}{\text{Throughput}_{\text{max}}} = \frac{\frac{\text{Power}_{\text{max}}}{\text{Throughput}_{\text{max}}}}{\text{Throughput}_{\text{max}}} = \frac{\text{Power}_{\text{max}}}{\text{Throughput}^2_{\text{max}}}$$

The Energy-Delay Product (EDP) [147] is another energy efficiency unit that is commonly used. However, the EDP does not capture parallelism if the delay is considered to be the critical path delay [58]. In this thesis energy efficiency is expressed as *energy per operation* or as ETR.

2.4.3 Energy-efficient Hardware

With energy efficiency it is the same as with scalability in the previous section 2.3, where there seems to be no universal unit capable to express the energy efficiency in an unbiased way under any condition. In order to understand the reason for the non-universality of energy-efficiency units, it is important to be familiar with the fundamental physical properties of processors. The basic building block of every processor is the transistor. A transistor, coined by John R. Pierce and derived from *transresistance* [219], is in its simplified form a switch, which controls the current flowing between two terminals by changing the resistance through a third terminal. These three terminals are commonly referred to as *source*, *drain*, and *gate*. By arranging and connecting transistors in certain ways [1], a logical node can be built to perform some functions. Carefully combining multiple logical nodes allows forming even more complex circuits until reaching the level of complexity of processors. Logical nodes and processors perform operations. These operations are considered energy-efficient if the energy consumed by executing the operations on the processor is optimal. Figuring out this optimal point of execution has been the quest of many electrical engineers and computer scientists over the past decades.

Performance and power consumption of a processor have a direct dependency. The power consumption of a processor can be decomposed as the sum of the dynamic and the static power consumption. Switching a transistor on and off is described by the dynamic component, while the static component describes power consumed by leakage currents. Performance depends on the frequency at which transistors switch on and off. The higher the frequency, the more performance can be achieved and the higher is also the power consumption. There are several variables in circuit design that can simultaneously minimize performance and power consumption by influencing the effective switched capacitance [58]. Other variables such as supply voltage and frequency are trading off delay and power consumption. Lowering the supply voltage reduces power consumption but increases the critical path delay. Reducing both frequency and supply voltage simultaneously saves power but increases delay. Power consumption and performance change linearly with frequency if the supply voltage kept constant.

2.4.4 Dynamic Voltage and Frequency Scaling

Dynamic Voltage and Frequency Scaling (DVFS) [305] is a mechanism which enables software to simultaneously scale voltage and frequency of an integrated circuit. Processor manufacturers determine sets of Operating Performance Points (OPPs) which are pairs of supply voltages and frequencies that guarantee sufficient margins under thermal worst case conditions for correct execution of operations. OSs provide interfaces to kernel modules which communicate with hardware to set supply voltage and frequency according to selected OPPs. CPUFreq [54] is the Linux kernel driver for DVFS. It defines so called governors which automatically adjust supply voltages and frequencies to pre-determined sets of condition. The basic and most common governors available on platforms are the *powersave*, *ondemand* and *performance* governors. Their names already imply the set of rules that will be applied by the governors. The powersave

[1] An exact description for connecting transistors to form logical circuits is subject of circuit design and out of scope of this thesis.

governor sets frequency and supply voltage to the lowest possible OPP in order to conserve power. This governor is primarily used by Linux when batteries of laptops are running low. The performance governor does the exact opposite of the powersave governors and sets frequency and supply voltage to the highest possible OPP. This governor is typically used in supercomputing where performance is needed for computationally solving larger sets of problems. The ondemand governor has a more complex set of rules which selects OPPs based on the current CPU load. If there is high demand for CPU cycles, then the governor will select higher OPPs depending on the load. Otherwise, if there are few programs executing sporadically, then the governor will select lower OPPs. Depending on the capability of platforms, there are many more governors available with various sets of rules that target different use cases. The three previously mentioned governors are used for this work and if not otherwise mentioned, the default governor is ondemand.

2.4.5 Undervolting

Platforms may provide configurations, interfaces or access to Model-Specific Registers (MSRs) for directly scaling supply voltages. Increasing supply voltage beyond nominal supply voltage is called overvolting while reducing supply voltage below nominal supply voltage is called undervolting. Supply voltages influence the rate at which capacitances in an integrated circuit can be charged and discharged, which determines the frequency processors can operate at. For processors overvoltage is needed on overclocked systems in order to sustain the higher than normal frequency. The power savings of DVFS are diminishing [179] with as the portion of dynamic power is getting smaller. Unlike frequency scaling, undervolting affects both dynamic and static power consumption of processors. Undervolting is in particular useful to reduce conservative voltage margins determined by processor manufacturers or identify save minimum supply voltages. Process variation describes various effects that influence the quality of the final processor during fabrication. Depending on the process variation, for example, one batch of processors can have higher maximum frequency than another batch or less voltage leakage. Process variation is taken into account by processor manufacturers when defining OPPs.

2.4.6 Power Meters and Power Distribution Units

Various power meters and Power Distribution Units (PDUs) were used for measuring the power consumption during evaluations. The choice between the use of a power meter or a PDU depends on the individual device. Power meters are typically designed for measuring the power consumption of a single device and offer higher resolution than PDUs. PDUs have multiple ports for powering devices which are configurable and they offer interfaces in order to switch on or off ports remotely. Power consumption of devices was recorded using the devices in the following subsections.

Table 2.1: Resolution and precision of power meters

Device	Measurements	Resolution	Precision
Alciom PowerSpy2	$50\,\mathrm{s}^{-1}$	10 mW	1 %
LINDY iPower Control 2x6M	$\sim 1\,\mathrm{s}^{-1}$	1 W	1.5 %
UniFi USW-48P-750	$\sim 8\,\mathrm{s}^{-1}$	n/a	n/a

Alciom PowerSpy2

The Alciom PowerSpy2[2] [19] is a power meter with a French socket. The device comprises a 4 GiB memory that allows to operate the power meter either in real-time mode or in logger mode. In real-time mode the PowerSpy streams the measured data directly over Bluetooth [82] to a listening device. Depending on the configured resolution at which measurements are taken, the device can log between 1 month and up to several decades worth of data. The PowerSpy measures the Root Mean Square (RMS) and peak-to-peak amplitude of voltage, current, and active power, the reactive and complex power, the power factor, the crest factor, the line frequency and the total harmonic distortion. Real-time mode gives access to the complete data set while logger mode only records the RMS and peak-to-peak amplitude of voltage, current and active power. Each measurement stands for an average value of 256 samples taken during one period. The PowerSpy can internally record up to 16.64 ksamples/s but is limited to a maximum of 50 measurements per second, *i.e.*, one measurement every 20 ms.

One major advantage of the PowerSpy is the open (but incomplete) documented Bluetooth protocol [127] used by the power meter. The open protocol specification allows to design custom implementations for various platforms besides the proprietary software suite that is provided by Alciom for the Microsoft Windows OS. Several software libraries for Python [198, 236] and Scala [80] have been available. This thesis relied on an older version of `powerspy.py` [198] that has been independently ported to Python version 3. Further adjustments were made to `powerspy.py` in order to permanently stream data at the highest possible frequency. The streamed data does not contain a timestamp and the `powerspy.py` script is taking a timestamp as the data is received over Bluetooth. An additional script was implemented which measures the Bluetooth round trip latency between the machine recording the data and the PowerSpy. Half of this round trip latency is subtracted from the timestamp in order to compensate for the transmission delay.

LINDY iPower Control 2x6M

The LINDY iPower Control 2x6M[3] [191] was a commercially available PDU. Two banks with each 6 IEC C13 [156] ports allow for a total of 12 devices to be connected to the PDU. Each bank has its own power supply with a maximum of 16 A and each port supports a maximum load of 10 A. The PDU is capable of measuring voltage, current, active power, reactive power, apparent power, frequency, phase angle and power factor. An Ethernet connection allows to control and monitor the PDU over a HTTP web interface. Additional protocols

[2] From here on referred to as PowerSpy.
[3] From here on referred to as the PDU.

supported by the PDU are SNMP, Syslog and SMTP. The power consumption is recorded over the HTTP web interface through a Python script that was initially written in late 2017 by Guillaume Fieni from the Spirals research group of University of Lille 1 and Inria. The web interface is updated roughly every second by requesting a query from URL path `/status.jsn.js?components=16384`. This query returns a JSON [102] object that contains the latest measurements for each port on the PDU. The Python script runs the same query from a remote machine and transforms the returned JSON object into CSV [277] and appends it to a file. JSON objects returned from the query do not contain a timestamp and instead timestamps are taken when the query request is posted. The script has been ported to Python 3.

UniFi PoE Switch

Ubiquiti Networks UniFi USW-48P-750[4] [293] is a network switch that supports 802.3af Power over Ethernet (PoE) [210]. The PoE status of connected Ethernet ports can be accessed through a local `telnet` connection over a protected API on the PoE switch. This interface is queried using an *ad hoc* `expect` [5] script that is based on an initial version written by Marcelo Pasin from the department of computer science of the University of Neuchâtel. The original script has been modified to continuously query the PoE switch for power consumption measurements. With the compute power of the PoE switch it is possible to query the PoE state about 8 times a second with the modified script. Timestamps are not included in the PoE status returned by the PoE switch and are added by the script upon data reception. In the PoE status information are included the negotiated power class level, the power drawn by the PoE device, PoE voltage and current as well as the temperature of the PoE unit in the PoE switch. The PoE switch is used in combination with Raspberry Pi PoE HATs [262] in order to power a cluster of 24 Raspberry Pis.

2.5 Security

This section defines properties and terms in the context of security and trusted computing as one of the challenges in this thesis for designing secure IoT services. Basic principles of symmetric-key, public-key and homomorphic encryption schemes are outlined in section 2.5.1 for securing services and in particular data from preying eyes or malicious attacks. Hardware-based cryptographic solutions (sections 2.5.2 to 2.5.3) provide additional security guarantees that make use of these previously mentioned basic principles. Secure systems [167], which are the foundation for implementing secure services, and which leverage hardware-based cryptographic solutions, are elaborated in more detail in this section and section 2.6. Furthermore, for these hardware-based solutions the trust model, various relevant trusted computing concepts and the GlobalPlatform API are explained. This section concludes with a terminology on service execution (section 2.5.4) that precisely defines the conditions satisfying the execution requirements.

[4]From here on referred to as PoE switch

2.5.1 Cryptography

Symmetric-key Encryption

Secure communication between senders and receivers over insecure physical transmission media, commonly referred to as *channels*, has been an important and challenging problem for a long time. *Ciphers* (*i.e.*, algorithms performing encryption and decryption) are making use of *confusion* and *diffusion* operations [278, 279]. Confusion operations obscure the tie between key and ciphertext typically by using substitution but remain susceptible to cryptoanalytical methods. Diffusion operations propagate a plaintext symbol over multiple ciphertext symbols in order to obfuscate statistical properties of the plaintext. Product ciphers are ciphers that chain confusion and diffusion operations to become more secure and increase their resistance to cryptoanalytical methods. Nowadays, most ciphers are product ciphers applying multiple rounds of confusion and diffusion operations.

Symmetric-key encryption is also known as *secret-key* or *single-key encryption*. After symmetric-key generation, the sender and receiver have to find a secure channel to share the symmetric key. The very same key is used for encrypting and decrypting messages which creates the problem of non-repudiation where both sender and receiver can make changes to messages and later deny them (*i.e.*, changes cannot be associated to an unique participant). In a scenario with multiple communication participants, each pair of sender and receiver needs to have a unique key. This makes key management and communication difficult and is better done using public-key encryption. The advantage of symmetric-key over public-key encryption schemes is the similarity of encryption and decryption functions. This allows for simplified implementations in hardware and software and makes symmetric-key encryption computationally less expensive. AES [92], Twofish [274] and Chacha [44] are well-known symmetric-key ciphers.

Public-key Encryption

Public-key encryption is based on the finding, that the key used by the person encrypting messages does not have to be secret as with symmetric-key encryption. The person receiving the ciphertext would instead have to be in possession of a secret key that allows decrypting ciphertexts. Such a public-encryption-private-decryption mechanism requires ciphers that make use of one-way functions. A one-way function is a function that can relatively easily be computed but whose inverse function is computationally infeasible. In this case, *relatively easily computable* means in the context of computational complexity that the function is at most of polynomial order whereas *computationally infeasible* means that the function is at least of exponential order. One-way functions have been identified for the domains of *integer factorization*, *discrete logarithm* and *elliptic curves*. In public-key encryption these problems are better known as Rivest-Shamir-Adleman (RSA) for prime factorization and Diffie-Hellman key exchange as well as Digital Signature Algorithm (DSA) with their elliptic curve equivalent for discrete logarithm.

Public-key encryption has several advantages over symmetric-key encryption. Secret keys can be established over insecure channels without the need for dedicated secure channels. Messages can be secured with digital signatures, as available in RSA, DSA or ECDSA, to avoid non-repudiation and guarantee message integrity. In order to ensure the authenticity of public keys, they can be derived from certificates and are therefore bound to an identity. The disadvantage of public-key encryption is that schemes are computationally more demanding than symmetric-key encryption schemes.

Homomorphic Encryption

Encryption preserves privacy and prevents other operations from manipulating encrypted data. Homomorphic encryption provides a solution to this problem, so that encrypted data can be offloaded and processed, for example in cloud data centers, without transferring the secret key and revealing the actual data. Many homomorphic encryption schemes have been proposed, ranging from schemes with a weak notion of privacy homomorphism called *partially homomorphic*, that can only perform a single function, to the strongest schemes called *fully homomorphic*, that can perform arbitrary functions. For example, cloud computing provides resources for offloading data and computation, but encryption, which is enforced by law for certain confidential data, nullifies all benefits of cloud computing [115]. A fully homomorphic encryption [114, 113] complies with legal restrictions on data privacy in public cloud environments and enables using public cloud resources for operating on encrypted data while guaranteeing confidentiality.

Homomorphic encryption can make use of both symmetric-key and public-key schemes and provides an evaluation operation. This operation evaluates functions on ciphertexts from a set of permitted functions [115] resulting in other ciphertexts, that when decrypted correspond to the result of applying the functions on the plaintext messages of ciphertexts. Evaluating any function that is not a part of the set of permitted functions, will result in ciphertexts that when decrypted do not contain any meaningful plaintext message. The implementation of evaluation operations requires that decrypting the resulting ciphertexts of evaluation operations take the same amount of computations (*i.e.*, have the same computational complexity) and have the same size as decrypting ciphertexts that were previously encrypted from plaintext messages. These additional requirements are referred to as *compact ciphertexts requirement* [115]. This means that the complexity of functions is independent of the size and the time needed to decrypt the resulting ciphertexts when applying evaluation operations. Only schemes that can evaluate any function and that satisfy the compact ciphertexts requirement are classified as fully homomorphic.

2.5.2 Trusted Computing Group

The Trusted Computing Group (TCG) [152] is creating work groups and specifications around the concept of Trusted Computing (TC), which is a combination of hardware and software-based security solutions. *Trust* is established by the behavior of the platform, which depends on the behavior of hardware and software [290]. In order to instantiate trust, platforms have to identify hardware and software components [290]. The OS has then to mediate [107] between these trusted hardware and software components to enable TC. The following paragraphs briefly outline some platform requirements that make TC possible.

Trusted Platform Module

The Trusted Platform Module (TPM) is an industry standard for a kind of vault embedded with a Root of Trust (RoT). TPMs provide various cryptographic subsystems, volatile memory, non-volatile memory, and Platform Configuration Registers (PCRs) as elements in order to collect and report behaviors of hardware and software components [291]. The following new TPM variants have emerged from the TPM 2.0 standard [9, 290]:

- Discrete TPM: separate, tamper resistant semiconductor package.
- Integrated TPM: integrated as a subsystem into another chip.
- Firmware TPM: software implementation in a TEE.
- Virtual TPM: accessible to VMs in cloud environments.
- Software TPM: implemented in software for emulation purposes.

Discrete TPMs are separate chips on platforms to which processors communicate over a bus. Older TPMs are susceptible to bus sniffing attacks with inexpensive Field-Programmable Gate Arrays (FPGAs) or more expensive logic analyzers by extracting keys in plain text from the bus when they are transmitted [200, 26]. *Integrated* TPMs are part of the processor and avoid peripheral buses for communication. The package of integrated TPMs might no longer be temper resistant since the chip fabrication process depends on the requirements of the processor which is incompatible with the security requirements of TPMs. *Firmware* TPMs are entirely implemented in software and their security depends on the software needed to operate TEEs. It is a weaker TPM variant than discrete and integrated TPMs, because it does not make use of dedicated hardware protection mechanisms. *Virtual* TPMs use the hypervisor for providing interfaces to physical TPMs on platforms. For this variant the security depends on the hypervisor which has to be trusted. Finally, *software* TPMs should only be used for emulation and testing purposes, since they are not protected and openly exposed to all kinds of attacks.

Trusted Computing Base

A system composed of hardware and software components can form a Trusted Computing Base (TCB). TPMs assist independent entities in establishing and validating TCBs, but are themselves not part part of the TCB. It is the responsibility of the TCB for preserving security policies of the system [290]. For this reason, TCBs need to have countermeasures in place to avoid being compromised by hardware and software components that are not part of the TCB. The TCB is typically measured in lines of code.

Root of Trust

Roots of Trust (RoTs) are fundamental elements that must be trusted, because any misbehavior is undetectable [290]. Certificates identify OEMs and provide necessary assurances for RoTs for the trustworthiness of their implementation. Platform certificates provide additional guarantees, that by platform manufacturer installed TPMs are trustworthy and compliant with TCG specifications.

TPMs provide four types of RoTs. The Core Root of Trust for Measurement (CRTM) resides in boot Read-Only Memory (ROM) and are the first instructions executed by processors when establishing a new chain of trust. Processors form the Root of Trust for Measurements (RTM) while they are controlled by the CRTM [290]. During measurements identities are communicated by extending PCRs located in the memory of the TPM that form the Root of Trust for Storage (RTS). PCRs hold hashes of measurements that are primarily used for validating logs of measurements upon request. Extending PCRs combines currently stored values in PCRs with hashes from measurements. The Root of Trust for Reporting (RTR) is responsible for reporting on the contents of the RTS. The identity of the RTR is a statistically unique (*i.e.*, with very high probability) and cryptographically verifiable asymmetric key known as Endorsement Key (EK). Reports are digitally signed hashes by the RTR that attest directly accessible contents in the RTS. A report contains current PCR values, audit logs and key properties [290]. The RTR reporting on the current PCR values is commonly called *quoting*. Platform certificates bind the RTR to the platform and provide the necessary proof in order to guarantee that the RTR is actually quoting PCR values of the platform.

Attestation

Attestation describes a process for demonstrating that software has been instantiated in an expected way on an intended platform. Software can be attested locally on the platform or remotely to a third party. Depending on the functionalities available on platforms, software is attested either by the CPU [24] or by the TPM [205]. The later will be considered for the remainder of this paragraph. Platform states are recorded in logs of software events and maintained in an integrity measurement architecture [270]. Every software event will be logged and recorded by the TPM as the platform is powered on and is booting up. For each software event in logs a measurement is produced by individually applying a cryptographic hash function and extended into a PCR. Attestations are composed of an untrusted event log and a signed quote from the TPM. Users submit challenges to TPMs which generate quotes in response containing a cryptographic nonce with a list of PCR values. In a first step users verify the authenticity of the signing key through the certificate chain before validating the signature of the quote. The untrusted event logs are then validated by recomputing the hash values in the PCRs and comparing them to the PCR values in the signed quote. This validation process allows users to make a trust decision based on the issuing Certification Authority (CA) of the signing key and the software state of the platform.

2.5.3 GlobalPlatform

The Open Mobile Terminal Platform (OMTP) was an industrial forum for mobile device standards that realized the need for trusted components and published in 2006 a first set of requirements for trusted environments (OMTP TR0). These requirements were revised in 2008 to define security requirements for mobile devices. A first standard for TEEs built on top of TR0 was released with OMTP TR1 [240]. After the transition of OMTP into WAC and the later integration into GSMA in 2010, GlobalPlatform has taken over the responsibility

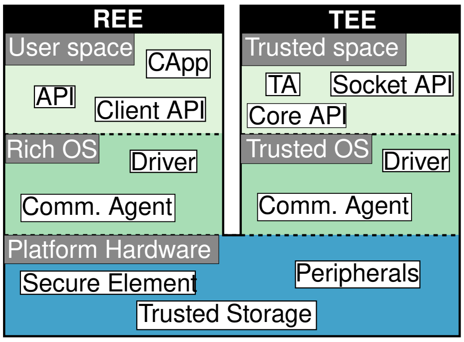


Figure 2.2: GlobalPlatform system architecture

on behalf of the industry to provides industry-wide specifications and standards for TEEs.

The GlobalPlatform TEE System Architecture [121] defines the necessary hardware and software requirements for operating TEEs as shown in Figure 2.2. Interfaces between user space and kernel space of normal worlds and TEEs are defined in standardized library APIs. The GlobalPlatform TEE Client API [117] defines the interface between Client Applications (CApps) and the rich OS, while the GlobalPlatform TEE Internal Core API [120] defines the interface between Trusted Applications (TAs) and the trusted OS. Besides these two primary interfaces, GlobalPlatform defines additional interfaces to interact, for example, with sockets through the GlobalPlatform TEE Sockets API [118]. Communication agents in rich and trusted OSs mediate data exchanges between normal worlds and TEEs.

Execution Environments

Execution Environments (EEs) provide basic hardware and software components for applications to support computing, memory management and access to peripherals [240]. These components are interfaced with either through APIs or Instruction Set Architectures (ISAs) depending on the implementation of the EE. Multiple EEs can be co-located on the same system, each with their own purposes and responsibilities. Communication mechanisms facilitate information exchanges and enable sharing resources between as well as within EEs. EEs that are generally designed for performance and provide many features as well as access to many peripherals, excluding trusted and secure resources, are called Rich Execution Environments (REEs).

EEs that satisfy security requirements according to specifications are called TEEs. TEEs can be authorized to access the Hardware Unique Key (HUK) [241], which can require Non-Disclosure Agreements (NDAs) between TEE providers and OEMs. The HUK resides in a shielded area of the TEE which is required to be resistant against several types of attacks [240]. Private keys provisioned with TEEs need to be confidentiality and integrity protected, while public keys have to be protected and validated for authenticity and integrity. Any code manipulating keys inside the TEE has to be integrity protected and verified. The verification is done at load time, but there are also solutions to verify the integrity of the TEE at runtime [221].

Secure storage allows storing resources securely, which can only be accessed from TEEs with dedicated storage keys. TEEs allowing execution of TAs have to provide mechanisms to install, verify and execute TAs. TAs need to be isolated from other TAs and EEs, so that code, data and keys belonging to TAs cannot be accessed. Data and keys used by TAs have to be protected and validated for authenticity and integrity. Secure services can be requested from TAs by CApps, which are executing outside of TEEs. Whenever resources are freed or TEEs are being terminated, any content has to be erased before these resources can be used again.

Root of Trust

The GlobalPlatform standard [119] defines the RoT similar to the TCG as a compute engine, code and data on the same platform providing security services. There does not exist an entity that can attest the state of the RoT. Two types of RoTs are distinguishes by GlobalPlatform which are *bootstrapped* and *non-bootstrapped* RoT. Bootstrapped RoTs are composed of several blocks of code with a first block called the Initial Root of Trust Component (iRoTc) (*i.e.*, CRTM). Any subsequent block of code is verified for integrity and authenticity, which creates an event, before being executed. Is a block of code unable to record the result of an event before executing the next block of code, then the RoT is extended and the block of code is called Extended Root of Trust Component (eRoTc). Execution is entering into a chain of trust if the result of an event is accessible in a shielded location to the subsequent blocks. The RoT is then said to be *bootstrapped* from the initial block to the independent block and forms a bootstrapped RoT. The idea behind bootstrapped RoTs is that blocks of code may come from different vendors and might not necessary have access to shielded locations for recording events. Non-bootstrapped RoTs consist of a single block of code that is created and provisioned by the platform manufacturer. Nine security services may be provided by RoTs which are authentication, authorization, confidentiality, identification, integrity, measurement, reporting, update and verification. These security services can be used by any other components such as CApp, TA or even different RoTs available on the platform such as TPMs or TEEs.

Trusted Storage

Trusted storage is a persistent memory area where data objects can be securely stored. Typically, such data objects are cryptographic keys or other confidential and sensitive data. Each data object is cryptographically protected and usually associated with an identifier and metadata in order to facilitate the retrieval of data objects. Hash trees [212, 213] are a convenient data structure for implementing trusted storage. Each node in the tree protects its child nodes through a cryptographic hash. Trusted storage can be located either on a dedicated storage device, such as an embedded MultiMediaCard (eMMC) [159] with some security properties or completely exposed to an insecure environment. EMMC is an embedded non-volatile memory that is available on many embedded, mobile and IoT devices. There are requirements that trusted storage has to provide a minimum level of protection against rollback attacks [120]. This protection includes a detection mechanism for rollback attacks independent from the location of the trusted storage.

2.5.4 The Different Forms of Execution

Execution of applications, programs and services can be provided by means of various hardware and software security mechanisms. Cloud providers offer services to users with levels of trust similar to that of their own devices. These services provide resources to users such as processor cycles, storage and networking while users provide data and code. Cloud providers have physical access to their services and users must protect their code and data in some form from the cloud provider. This scenario is the opposite of sandboxing where the confidentiality and integrity of the code and data of users has to be protected from the host. Cloud providers control resource allocation and have an interest to protect their infrastructure from malicious users through various protection mechanisms such as isolation of processes or sandboxing.

Isolated execution [41, 207, 242, 296] is a protection mechanism that maintains the confidentiality and integrity of code and data. Software-based solutions rely on the one hand on trusting higher privileged code to provide confidentiality and integrity guarantees while on the other hand hardware-based solutions only add specific code to the TCB. *Shielded execution* [41] is a generalization for specific implementations [69, 74, 145, 183, 204, 205, 248] using isolation mechanisms in order to provide higher security guarantees. Systems are oblivious to the execution of shielded programs and can only observe their inputs and outputs. The execution of shielded programs cannot be influenced by systems, except through DoS, and the results of the executions on identical platforms are always the same. *Confidential execution* is what users need for deploying their services in the cloud. Attestation and secure provisioning mechanisms need to extend shielded execution for providing additional integrity and confidentiality guarantees, that allow users to remotely verify the state of their deployed services and decide whether to trust them before provisioning any secrets.

2.5.5 Attacks

Deployed devices, in particular those connected to the Internet, are constantly exposed to various threats. These threats manifest in physical or remote attacks in order to compromise systems or gain access to confidential data. There exist various attacks that are all classified by the general principle they are trying to achieve. For example, DoS attacks prevent services from executing and there are many ways in which attacks can achieve the DoS. Attacks are primarily exploiting bugs and flaws in hard- and software, however, physical and especially side-channel attacks make use of information gained during the execution of the target system. There are approaches for mitigating or preventing certain attacks, but absolute security is not attainable. TEEs are one approach that is relying on a combination of hard- and software techniques for providing security guarantees against specific attacks. The following subsection briefly describe a selection of attacks that appear within the context of this thesis.

Denial-of-service Attacks

Denial-of-Service (DoS) is a type of attack that requires little effort [275] to deploy and prevents services from executing. The attack is typically targeting the resources available to services and is trying to deplete them in some way. This kind of attack was initially used in networks as SYN flooding [81, 275] to prevent nodes from responding to requests by filling up memory with open but unused connection. DoS is not limited to networks and can be used against any service, provided there are some complex interactions in order to exploit hardware failures and software bugs or exhaust resources [309]. Distributed Denial-of-Service (DDoS) is a network-based attack using multiple nodes to coordinate a DoS attack that is difficult to identify because of the different request sources. Preventing DoS attacks is challenging because the attack needs to be identified first before it can be mitigated. The identification has to distinguish between legitimate request and malicious requests. In some instances services can also be subject to unintentional DoS due to some popular target made available by the service that is highly requested.

Person-in-the-Middle Attacks

Person-in-the-Middle (PITM) attacks, also known as Man-in-the-Middle (MITM) attacks, describe scenarios where an attacker can craft messages and inject them into channels or intercept messages that are exchanged over channels to modify them. By actively taking part in the message exchange over channels and influencing it, an attacker can defeat the protocol. Any asymmetric scheme can be defeated by PITM attacks unless the public-keys are protected by certificates [243]. The attacker substitutes the public keys exchanged over the channel between two entities by their own public keys. This is for example how the public-key Needham-Schroeder protocol [232] can be exploited [193]. Authentication can be provided in the form of certificates, by signing the public key and the identity of the entity with the private key. Such certificates are typically issued by CAs.

Replay Attacks

Replay attacks are specific kinds of PITM attacks, in which messages are being intercepted and resent to the receiver in order to access resources intended for the original sender. *Freshness* is another term that is used for this concept, which emphasizes that the receiver should only receive new (*i.e.*, fresh) messages and not old (*i.e.*, rotten) messages. The symmetric-key Needham-Schroeder protocol is vulnerable to replay attacks in case of compromised communication keys [95]. Timestamps were introduced to prevent replay attacks and replace the two-step handshake in the Needham-Schroeder protocol. Other approaches to prevent replay attacks include nonces combined with MACs, one-time passwords and randomly chosen session IDs. TLS prevents replay attacks by computing a MAC that includes MAC key, sequence number, message length, message contents and two fixed character strings [97].

Downgrade and Version Rollback Attacks

Communication protocols, such as SSL and its successor TLS, are susceptible to downgrade attacks, also called version rollback attacks. These attacks exploit design flaws in protocols to fall back to older protocol versions, that exist for compatibility reasons, but might be vulnerable to other attacks. In case of the now deprecated SSL protocol [39], downgrade attacks allowed to weaken confidentiality guarantees by using weaker *export-grade* cipher suites [300] (due to US export regulations before 2000). RSA keys of export-grade key sizes were already factorized in 1999 [62] and are therefore considered insecure. This flaw was addressed in SSL version 3.0 by authenticating all handshake protocol messages with the master secret, but also introduced new design flaws. Despite the known flaw in the SSL protocol, TLS was vulnerable to the FREAK attack [45] exploiting a bug in TLS implementations to downgrade RSA key exchanges to export-grade keys, and to the Logjam flaw [11] that lets PITM attacks downgrade connections to export-grade Diffie-Hellman key exchanges. In order to prevent this type of attack, the best approach is to remove legacy code (previous protocol versions) and backward compatibility from the code base.

Rollback Attacks

The problem of TEEs is they cannot guarantee that data external to the TEE has not been rolled back, *i.e.*, reverted to a correct but earlier state or an older version beyond their lifetime [41]. In the instance where TEEs crash as a result of malicious attacks or bugs in the TCB of the TEE, their internal state which was stored in volatile memory is lost. When a new TEE is then instantiated, it can verify if data is consistent, but not if this was the state or version of the data before the TEE crashed. Confidentiality and integrity guarantees of the state are insufficient for preventing rollback attacks, and a continuity guarantee [248] has to be introduced. In order to provide such continuity guarantees, that data has not been rolled back, secure non-volatile storage is necessary to recover the state of a TEE after a crash.

Side-Channel Attacks

Side-channel attacks exploit effects during the execution of cryptographic operations in order to leak confidential information. There are various types of side-channel attack such as timing [170, 313] or power [169, 288] attacks. These types of side-channel attacks exploit time differences for cryptographic operations, power consumed or acoustic signals generated by the cryptographic processor during cryptographic operations. All these effects are physical properties of correctly implemented cryptographic processors and are not related to faults in hardware. Cryptographic processors need to implement additional measures to make cryptographic operations indistinguishable from each other or randomly change the frequency to introduce noise in the power consumption in order to prevent such attacks.

2.6 Trusted Execution Environments

The TEE section describes the hardware and software architecture and standards for using TrustZone with OP-TEE. TrustZone and OP-TEE are being used as reference TEEs in chapters 3 to 4 for shielding IoT services. Intel SGX [24, 144, 207] and AMD SEV [164] are briefly outline as the latest TEEs on the x86 architecture available on mobile, desktop and cloud platforms. TEEs for other architectures include IBM Secure Service Containers [153] and IBM Secure Execution [161] as well as RISC-V MultiZone [141].

2.6.1 ARM TrustZone

TrustZone is an optional set of hardware-enforced security extensions that in combination with secure software is forming a TEE, which is space efficient with only little impact on power consumption and performance [21]. With TrustZone OEMs can seamlessly integrate security into their hard- and software solutions for creating trusted platforms. The security rooted in hardware can be extended across all layers ranging from firmware all the way to applications. The isolation is based on dualistic worlds, in which the *secure world* includes the TEE wherein the *non-secure world* or *normal world* includes everything else. The secure world provides data processing capabilities and access to memory as well as peripherals at full bus-bandwidth [21]. Unlike secure modules, TrustZone can be further customized through secure software. Additional security solutions can be build on top of the TrustZone foundation, such as hardware encryption engines for memory or storage encryption.

As for all TEE technologies, TrustZone does not provide absolute security, therefore continuous architectural improvements are made to protect against attacks. Each bus transaction includes a Non-Secure (NS) bit indicating the world to which transaction belong to. CPU and memory are logically split into normal and secure world by the NS bit whereat the CPU is used in a time-shared manner. Besides the common ELs, another level with the highest privileges (*i.e.*, EL3) is introduced for the *secure monitor* only, which exists in the secure world. The secure monitor is responsible for switching between the two worlds whenever secure monitor interrupts are received as a result of executing Secure Monitor Call (SMC) instructions or receiving hardware exceptions [29]. When the worlds are being switched, the secure monitor saves the state of the currently executing world and restores the state of the world that is being switched to before returning from exception into the restored world.

TrustZone proposes three mechanisms for adding code to access peripherals from the secure world. Either borrow the driver from the normal world through an interface or move the driver into the secure world and provide an interface in the normal world kernel. Another mechanism is to simply duplicate the code and make it available to both worlds, in conjunction with establishing a protocol for regulating access to the shared peripheral. The last mechanism only works for peripherals generating interrupts, such as keyboards, mice, and touchscreens, where interrupts can be redirected to the secure world without involving the normal world interrupt handler. The TrustZone secure software includes the secure monitor for switching between worlds, a secure kernel with secure drivers to form the trusted OS, a secure boot loader to start the trusted OS and secure services running as TAs. Linaro, an organization engineering

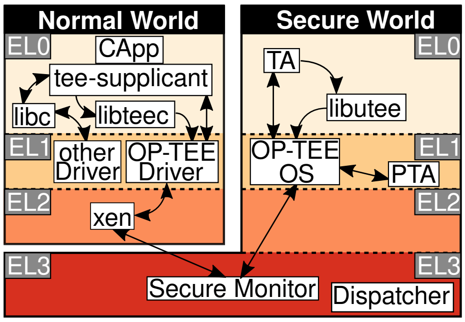


Figure 2.3: TrustZone exception levels and how OP-TEE maps and connects its components to them.

software for the ARM architecture, is currently maintaining the secure software, such as TrustedFirmware and OP-TEE.

2.6.2 Open Portable Trusted Execution Environment

The Open Portable Trusted Execution Environment (OP-TEE) is a TEE implementation of the standards and specifications published by GlobalPlatform, that is based on TrustZone. OP-TEE provides support for various ARM platforms including emulation with QEMU. In particular support for QEMU is interesting for developers, which allows to quickly test and evaluate newly implemented features on TrustZone platforms in an inexpensive way and with very few drawbacks compared to TrustZone hardware.

Architecture

The architecture of OP-TEE is summarized in Figure 2.3 and shows the call graph. OP-TEE relies on the Trusted Firmware [188] as foundation, which provides the dispatcher and the secure monitor that run in secure EL3. When the device is booting up, the firmware allocates memory regions to the secure world that are used by OP-TEE. Once all memory regions are setup, the dispatcher will initiate the next boot stage. If the ARM processor is compatible with ARMv8.4 it will boot the secure partition manager at secure EL2. Otherwise, the dispatcher will skip to the next boot stage using specific code to load OP-TEE OS. OP-TEE OS runs at secure EL1 and consists of the core of the secure kernel and Pseudo Trusted Applications (PTAs). TAs are executed at EL0 where they have access to `libutee` which provides an implementation of the GlobalPlatform Internal Core APIs and the TEE Sockets API. TAs and secure storage are either stored on eMMC if available or in the REE file system. At EL2 in normal world can optionally be running a hypervisor with support for OP-TEE. At EL1 runs a Linux kernel with the OP-TEE kernel driver. The OP-TEE driver in the Linux kernel makes it possible for Linux distributions to interact between both OSs. The OP-TEE driver is in particular needed for allocating shared memory, such that applications running in the REE can provide data to TAs. Whenever OP-TEE OS has to request services from the normal world, it contacts the `tee-supplicant` running as a daemon at EL0 through Remote Procedure Calls (RPCs). This is necessary for loading TAs or for retrieving data objects from the secure storage that are stored in the REE file system. By default network support is also provided from the REE where OP-TEE is borrowing the network stack of the REE to provide network functionalities to TAs. In this case all network system calls are handled by the `tee-supplicant` through a series of RPCs from the TEE. When requesting services from the TEE, applications can invoke system call through `libteec` which implements the GlobalPlatform TEE Client API.

Trusted Applications

TAs are modified ELF binaries with constraint memory resources and a local heap. In OP-TEE, TAs are single-threaded processes that are executed by the trusted OS OP-TEE OS in secure exception level 0 (secure EL0). They use *universally unique identifiers* (UUID) as file names and can be optionally encrypted. TAs can be stored either in the REE filesystem, in secure storage or be directly linked into the TEE core. Each TA stored in the REE has to be cryptographically signed at compile time with a private RSA key to provide authenticity and integrity protection.[5] This is important because TAs stored in the REE are exposed to a potentially compromised environment. When TAs are loaded by OP-TEE OS from the REE, a version check is performed against a list of TA versions OP-TEE OS maintains in secure storage to prevent any downgrade attack. OP-TEE OS then verifies the signature of the TA using the public key contained within OP-TEE OS before instantiating the process. *Secure storage* TAs are as, the name implies, stored in secure storage. In fact only meta data is stored in a directory file which is itself stored encrypted and integrity protected in secure storage. The actual secure storage TA is stored encrypted with AES-GCM as a file in the REE filesystem. All information to decrypt secure storage TAs can be found in the meta data. *Early* TAs are directly linked into the TEE core and execute before the `tee-supplicant` is available. Besides being stored inside the TEE core and executing earlier than any other TA, there are no other differences to TAs from the REE filesystem. PTAs are similar to Linux kernel modules and provide distinct secure services. Despite the TA in their name, these are not TAs and cannot be interfaced with through the GlobalPlatform TA Client API. PTAs are statically linked into OP-TEE OS and use its internal core API as interfaces. Therefore, PTAs execute at a higher exception level than TAs and are more privileged.

Memory Configuration

TAs in OP-TEE have a private heap which is created by the TEE core when TAs are instantiated. The size of the stack and the heap is determined at build time and defined by the `TA_STACK_SIZE` and `TA_DATA_SIZE` macros. By default, the stack is set to 2 KiB and the heap to 32 KiB. There are several limiting factors that prevent arbitrary large TA stack and heap sizes. The default total memory allocated for the TEE is limited to 32 MiB. This memory is divided into 1 MiB of `TEE_RAM` for the core, 4 MiB of `SHMEM` for non-secure shared memory and 27 MiB of memory available to TAs. In theory one TA could use 27 MiB of memory but

[5] It should be pointed out that [123] mistakenly claims that OP-TEE does not verify the integrity of TAs. This only applies to TAs not stored in the REE file system.

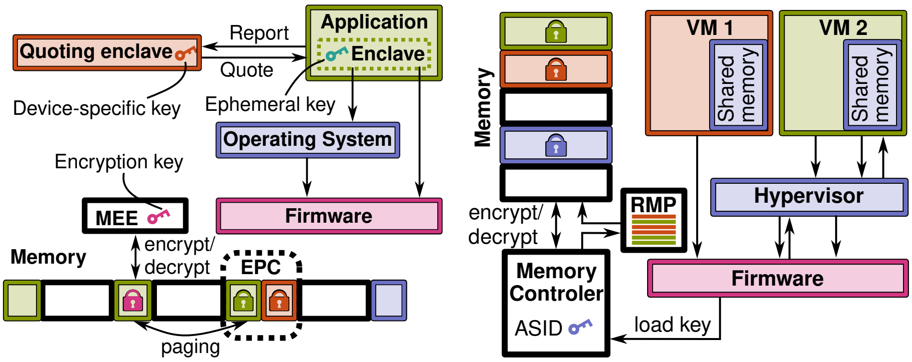
in practice Executable and Linkable Format (ELF) [289] sections of TAs in OP-TEE are mapped to page translation tables with 2 MiB size. Therefore, it is not possible to allocate more than 2 MiB of heap unless the configuration of OP-TEE is changed. These changes involve hardcoding the number of active page tables `PGT_CACHE_SIZE` and adjusting the number of concurrently executing TAs with `CFG_NUM_THREADS`. The same restrictions also apply when more than 2 MiB of memory need to be shared.

Secure Storage

Secure storage is an implementation of trusted storage described in the GlobalPlatform TEE Internal Core API [120]. The secure storage provides confidentiality and integrity guarantees for stored data objects. Any operation performed on the secure storage has to be atomic, meaning that the operation completes successfully only if it is not interrupted. OP-TEE can provide secure storage in the REE filesystem or in a replay protected memory block of eMMC [159]. Data objects that are to be stored in secure storage are encrypted by the TEE file system and forwarded to the `tee-supplicant`. The `tee-supplicant` then adds the encrypted data object to the secure storage by interfacing with the Linux file system. Retrieving data objects from the secure storage works in a similar way by passing through the `tee-supplicant`. Three types of storage keys are managed by the key manager, which is part of the TEE file system. The SSK is a HMAC using SHA256 as hashing algorithm and derived from the HUK concatenated with the chip ID and a static string. At boot time the SSK is generated and stored in secure memory. Each TA has its own storage key that is a HMAC using SHA256 and derived from the SSK and the UUID of the TA. The Trusted Application Storage Key (TSK) protects the File Encryption Key (FEK), which is used for encrypting and decrypting meta and block files. The key manager generates the FEK from the pseudo random number generator. In order to provide integrity protection, the secure storage uses a hash tree, in which the parent node protects its child nodes. OP-TEE provides interfaces to read the HUK but does not provide any implementation for it. The difficulty of accessing HUKs is discussed in section 3.6.

Shared Memory

OP-TEE, which is compliant with GlobalPlatform APIs, provides three types of shared memory. Whole and partial shared memory are shared memory that has to be requested by the application running in the REE and is allocated by the Linux kernel driver. Whole shared memory corresponds to an entire memory region allocated by the Linux kernel driver, while partial shared memory is part of a whole shared memory with a specified offset. [117] For better performance only dynamically allocated and physically non-contiguous shared memory has to be registered with OP-TEE, whereas physically contiguous shared memory is already known to OP-TEE OS. Temporarily shared memory is a memory buffer allocated by the application in the REE which OP-TEE then maps as shared memory for the duration of the API call [117]. This type of shared memory is dynamically allocated and can benefit from better performance by registering it.



(a) Basic principle of SGX (b) Basic principle of SEV

Figure 2.4: Basic principles of further TEEs

Interface Sockets

`libutee` provides through the GlobalPlatform TEE Socket API a network interface to TAs. Interface sockets are handled inside OP-TEE OS by the socket PTA and their lifecycle is bound to the TA session. The socket PTA translates the secure system calls into RPCs for the `tee-supplicant` and executes SMC instructions to switch into the normal world. In the normal world, the `tee-supplicant` is constantly pulling the OP-TEE kernel driver for new service requests from the TEE in from of RPCs. Upon receiving RPCs from the socket PTA, the `tee-supplicant` extracts parameters as well as data and executes the corresponding POSIX socket function. The implementation in the rich OS handles the socket system calls and interacts with the Network Interface Controller (NIC) driver for sending and receiving data over the network. Any data that is being sent or received over sockets by TAs has to traverse all exception level in the secure and non-secure world. In total between 5 to 8 context switches are necessary, from secure EL0 up to EL3 and down from normal EL2 to EL0 before going back up to EL2, depending on any hypervisors used in normal and secure world, before data is available. OP-TEE did not provide an implementation for the TLS protocol at the time, meaning that secure connections between TAs could not be established through the TEE Sockets API.

2.6.3 Further Trusted Execution Environments

TrustZone together with Software Guard Extensions (SGX) and Secure Encrypted Virtualization (SEV) are the three most popular TEEs. SGX and SEV are briefly described in this subsection for comparison to TrustZone. Each TEE provides various security functionalities and is targeting different platforms.

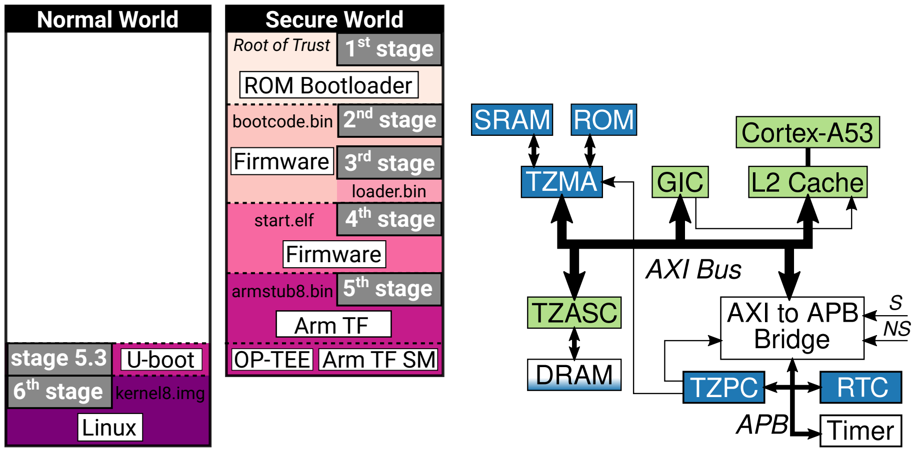
Intel Software Guard Extensions

Intel SGX [24, 144, 160, 206, 207] is a TEE with strong security guarantees that was first made commerically available with the 6$^{\text{th}}$ generation Intel Core processors of the Skylake microarchitecture [85, 86]. SGX provides instructions implemented with XuCode [88] to securely deploy software containers, called *enclaves*, locally or remotely over the Internet. The SGX TCB only includes the processor, firmware to operate the processor and software running inside the enclave. Software can be divided into an untrusted part and an enclave in order to shield only the security critical code with the help of the Software Development Kit (SDK) [83]. The Enclave Page Cache (EPC) designates a memory region that is used as shielded location for enclave pages. During the launch of enclaves SGX records logs of the content of enclaves and appends them to reports. After the launch it is no longer possible for any unauthorized software, including OS, hypervisor, BIOS and firmware, to access memory regions of enclaves or append additional measurements to their logs. Once enclaves have successfully launched, they enter the attestation phase and are ready to be provisioned with confidential data. For remote attestation a *quoting enclave* is provided with access to the device-specific private key that is used to sign reports. If the EPC is exhausted at runtime, enclave pages are paged out to regular memory after being encrypted by the Memory Encryption Engine (MEE) and integrity protected. Hardware enforced access control policies protect code, data and stack of enclaves from attacks at runtime.

AMD Memory Encryption

AMD have introduced their own TEE [164] with the Zen microarchitecture [12, 180]. The TEE is primarily targeting cloud data centers and integrates two novel memory encryption technologies called Secure Memory Encryption (SME) and SEV. Confidentiality of data is no longer guaranteed when it is loaded into memory of collocated cloud instances, since data is accessible in memory as plaintext. This is even more problematic with non-volatile memory technologies such as NVRAM, where malicious administrators can access unencrypted data by physically removing the DIMM and plugging it into another system. The memory encryption technology is integrated into the CPU architecture and allows to use unmodified applications. Memory controllers feature an AES engine that can be programmed with multiple keys by firmware running on the Secure Processor (SP), an embedded ARM Cortex-A5 microprocessor. Any data stored to memory is encrypted with a 128-bit key and decrypted with the same key when loaded from memory. The encryption of memory pages is controlled by the page tables managed by the OS and hypervisor. With enabled memory encryption, bit 47 in the physical address, called the C-bit, indicates if the memory page has to traverse the AES engine.

SME is using a single key for either full memory encryption or partial memory encryption. SEV integrates the memory encryption technology with the AMD-V [14] virtualization architecture for supporting encrypted lightweight virtualization containers and VMs. Code and data of each VM and the hypervisor also are provided with a tag of its VM Address Space Identifier (ASID) inside the SoC restricting its usage to the owner with the same ASID and protecting from any unauthorized access. Encrypted State (ES) [163] revises a flaw [139] in the implementation of SEV where the state of the registers is not cleared and stored in plaintext to an unencrypted location in the hypervisor. Any sensitive data that was processed and stored in registers by the VM during the interrupt is leaking to the hypervisor. Secure Nested Paging (SNP) [13] is the latest extension to SEV providing memory integrity protection against replay attacks. A system-wide Reverse Map Table (RMP) records the owner of every memory page and its mapped memory location in order to enforce access



(a) Boot sequence

(b) ARM AMBA and TrustZone with secure (blue) and configurable or TrustZone-aware (green) components.

Figure 2.5: Raspberry Pi model 3B and 3B+ specific boot sequence and TrustZone hardware implementation.

control.

2.7 Raspberry Pi

The Raspberry Pi is a credit card sized embedded platform that features an ARM Cortex-A microprocessor on a Broadcom SoC with numerous peripherals. Raspberry Pis are designed by the Raspberry Pi Foundation and exist in different versions and from factors, of which model B is exclusively used for the current work. The Broadcom SoC of the model B includes a quad-core ARM Cortex-A microprocessor bundled with a VideoCore [53] GPU. Memory is available as Low-Power Double Data Rate (LPDDR) Synchronous Dynamic Random Access Memory (SDRAM) and, with newer Raspberry Pi models, can be obtained in various sizes. Peripherals include display and camera connectors, HDMI, USB, Ethernet, a Wi-Fi and Bluetooth module, an audio jack, 40 GPIO pins and a microSD-card slot. The Raspberry Pi provides a very popular platform because of the rich software support compared to other embedded and IoT platforms. The available peripherals and the software support make it an ideal IoT prototyping platform. Raspberry Pi models 3B, 3B+ and 4B were used throughout this thesis.

2.7.1 Boot Sequence

When Raspberry Pis are powered on, they immediately start up in secure world and the first component enabled is the VideoCore GPU. Inside the GPU is a real-time OS running which handles the early boot process, licensing of video codecs and power management. This real-time OS forms the RoT of every Raspberry Pi and is the first boot stage of the boot sequence. Figure 2.5a shows the boot stages Raspberry Pi models 3B and 3B+ runs through when powering on and booting an OS. The boot sequence for the Raspberry Pi 4B is similar but deviates from previous models because the firmware is stored on a SPI EEPROM. The remaining subsection describes the boot sequence for the models 3B and 3B+. The real-time OS then loads the `bootcode.bin` file from the microSD card into L2 cache and jumps to it. This binary contains firmware and forms the second and third boot stages, while there used to be a `loader.bin` file for the third stage that is now obsolete. The second stage enables the SDRAM and loads the `start.elf` file before running it. Configuration files are parsed in the fourth stage and the firmware loads the `armstub8.bin` and `kernel8.bin` files into memory for preparing to release the reset signal on the ARM CPU. In the fifth stage the reset signal is released and the ARM CPU has two different ways to boot up the system. Either it starts executing `armstub8.bin` if available or boots the OS in `kernel8.bin`. `armstub8.bin` prepares stages 5.1 and 5.2 to load the ARM Trusted Firmware and dispatch the trusted OS. When the trusted OS has booted the CPU switches into the normal world and continues with stage 5.3 and the normal world boot loader such as U-boot. If `armstub8.bin` is not available, then stage 5 is skipped and the CPU begins booting the OS in the normal world.

2.7.2 Performance and Thermal Design

The firmware can be adjusted through the configuration file for modifying various parameters such as CPU and GPU clock frequency, supply voltage, soft limit temperature and many more. Supply voltages on the platform can only be changed in steps of 25 mV, which are defined as overvoltage levels in the configuration file. The default voltage offset is defined as overvoltage level 0 and any higher or lower overvoltage level applies a voltage offset that corresponds to a multiple of 25 mV. For example, an overvoltage level of -2 applies a voltage offset of -50 mV on top of the base supply voltage. The minimum and maximum overvoltage levels are -16 and 8. Setting an overvoltage values beyond 6 also requires setting the `force_turbo` option which enforces permanent turbo frequencies. If any other overvoltage option is chosen, the firmware will set a flag which voids the warranty of the Raspberry Pi because of damaging overclocking settings. Raspberry Pis feature an undocumented Adaptive Voltage Scaling (AVS) system which determines the base supply voltage when the platform is powered on and dynamically adjusts supply voltage based on various conditions as well as a feedback loop of sensor data. Most Raspberry Pis have no more than two OPPs - one for minimum and one for maximum performance. These OPPs can be force set with the Linux CPUFreq [54] *powersave* and *performance* governors. The *ondemand* governor will operate at the minimum OPP when the core is idling and switch to the maximum OPP when the CPU load crosses a certain threshold.

Models 3B and the 3B+ share the same Broadcom BCM2837[6] SoC but use a different thermal design. The 3B has a plastic package, while the 3B+ has a package with a metal shield and uses a MaxLinear MxL7704 [203] power management chip. These thermal changes allow the 3B+ to run at a 200 MHz higher clock frequency. The firmware for the 3B+ introduces a soft limit at

[6]Neither Broadcom nor the Raspberry Pi Foundation have made data sheets publicly available for the BCM2836, BCM2837, BCM2837B0 and BCM2711 SoCs used on Raspberry Pis.

70 °C which throttles the CPU and reduces the clock frequency to 1.2 GHz as on the 3B and also the supply voltage, such that many use cases never reach the hard limit throttle at 80 °C [294]. The 4B uses a very similar thermal design to the 3B+.

2.7.3 TrustZone Support on the Raspberry Pi

The Broadcom SoC lacks support for cryptographic acceleration instructions and does not provide an ARM TrustZone Protection Controller (TZPC), an ARM TrustZone Address Spec Controller (TZASC), an ARM Generic Interrupt Controller or any other proprietary security control interfaces on the AMBA bus [34]. Only the ARM Cortex-A cores are TrustZone aware on the Raspberry Pi model B, which is sufficient for minimal support of TrustZone functionalities. Figure 2.5b shows a block diagram of various SoC components connected to the AMBA bus and how TrustZone integrates with these components. Components can directly be accessed over the AXI bus since most TrustZone components are missing on the Raspberry Pi. On the Raspberry Pi there is no isolation of memory between normal world and secure world, because the TZASC is missing. The model B is also not equipped with an on-chip memory in form of SRAM or eMMC for secure storage. This means that it is possible to access any secure memory from the normal world and potentially extract confidential and sensitive information at runtime. Consequently, the Raspberry Pi should be considered as an educational platform that introduces the concepts of TrustZone as a TEE.

2.8 Equipment

All devices used in this thesis belong to the computer science department of the University of Neuchâtel. In recent years a tradition was introduced to name devices after Swiss mountains. Mountain names are convenient for substituting difficult to read alphanumeric model names of devices that are hard to be remembered. This section summarizes all specifications of the devices used throughout this thesis and refers in the following chapters only to their mountain pseudonym listed in table 2.2. Specifications of the CPUs are mentioned in table 2.3. The specifications of the memory hierarchy for each device are detailed in table 2.4 and table 2.5.

Table 2.2: Devices used for evaluation in this thesis

Pseudonym	Manufacturer	Model	Year
Chaumont	Raspberry Pi	4B	2020
Gibloux	Raspberry Pi	3B	2016
Kaiseregg	Hewlett Packard	ProLiant SE1102	2008
Pilatus	Supermicro	5019S-M2	2016
Vully	Raspberry Pi	3B+	2018

Table 2.3: CPU specifications of devices

Name	Manufacturer	Family	Model	Cores	Clock	Sockets
Chaumont	Broadcom	Cortex-A72	BCM2711	4	1.5 GHz	1
Gibloux	Broadcom	Cortex-A53	BCM2837	4	1.2 GHz	1
Kaiseregg	Intel	Xeon	L5420	4	2.5 GHz	2
Pilatus	Intel	Xeon	E3-1275 v6	4	3.8 GHz	1
Vully	Broadcom	Cortex-A53	BCM2837B0	4	1.4 GHz	1

Table 2.4: Cache specification of devices

Name	L1i	L1d	L2	L3
Chaumont	4× 48 kB	4× 32 kB	1 MB	n/a
Gibloux	4× 16 kB	4× 16 kB	512 KiB	n/a
Kaiseregg	4× 32 KiB	4× 32 KiB	2× 6 MiB	n/a
Pilatus	4× 32 KiB	4× 32 KiB	4× 256 KiB	8 MiB
Vully	4× 16 kB	4× 16 kB	512 KiB	n/a

Table 2.5: Memory specification of devices

Name	RAM	Gen	Transfer rate
Chaumont	4 GiB	LPDDR4	1866 MT/s
Gibloux	1 GiB	LPDDR2	800 MT/s
Kaiseregg	2× 4 GiB	DDR2	667 MT/s
Pilatus	16 GiB	DDR4	2400 MT/s
Vully	1 GiB	LPDDR2	800 MT/s

3 Understanding the Impact of Shielding on IoT Services

3.1 Introduction

The lack of trust in cloud providers and their infrastructure has greatly limited the deployment of IoT services on public clouds. Cryptographic approaches may address these privacy concerns but they also introduce new challenges. For example, key management and provisioning are increasing the complexity and development time of services. Furthermore, data access and processing performance are reduced because of the introduction of a cryptographic overhead. Decreasing performance results in prolonged runtime for operations that directly affect the energy consumption and costs of services. Deploying secure IoT services appears to require more resources and is considered an expensive challenge. In public clouds the growing availability of TEEs as hardware-based solutions to these challenges enable promising system architectures for secure IoT services. SGX and SEV are two TEE technologies that are primarily available on server-grade x86 architectures used in cloud data centers. Table 3.1 gives an overview of available TEE instances for a selection of cloud providers.

The IoT and mobile ecosystem is diverse and rich in hardware and software solutions. Currently, ARM processors and microcontrollers hold the majority of the market share in this ecosystem with their power-efficient designs, but new and open architectures such as RISC-V are on the rise with an ever growing market share. Since version 6 of the ARM architecture, selected processors and microcontrollers include an optional set of security hardware features and instructions called TrustZone. TrustZone is the oldest TEE technology initially designed for embedded and mobile devices, that has also found its way into the server market with recent server-grade ARM processors. Changes made to the architecture also introduce new security features, such as ARMv8.4 enabling virtualization in the secure world [31]. Secure partition managers allow simultaneous use of multiple trusted OSs as well as platform firmware and provide better isolation mechanisms for resources.

Many TEEs and frameworks have emerged for various ARM platforms with the standardization of TrustZone-based TEEs. These TEEs and frameworks provide developers necessary tools and APIs for building TAs by abstracting the complexity of the underlying hardware and software components. The standards address primarily higher level interfaces and APIs as well as general hardware resource requirements, but leave open many details in particular with respect to accessing peripheral resources securely for implementing advanced TAs. As a result, TAs remain specific to their trusted OS, because libraries and access to

Table 3.1: TEE availability by cloud provider

TEE	Alibaba	AWS	Azure	GCP	IBM	Oracle	Tencent
SGX	✔ [20]	✘	✔ [217, 218]	✘	✔ [154]	✘	✔ [194]
SEV	✘	✘	● [216]	✔ [122]	✔ [155]	✘	✘
TZ	✘	✘	✘	✘	✘	✘	✘

resources provided by the trusted OS are either platform or vendor specific for intellectual property reasons. Even dispatching trusted OSs requires OS-specific code in the TrustZone firmware. Portability of TAs across different trusted OSs is limited and requires several variants of a single TA for deployment on multiple trusted OSs. Considering existing standards and APIs, development of portable TAs appears as an unnecessary cost and time investment.

During the implementation of secure services developers are facing performance bottlenecks that are hidden by layers of abstraction. Identifying and resolving these bottlenecks is crucial for IoT services, since many IoT devices have limited energy resources and depend on high performance or efficient implementations to reduce energy costs. Little is known about the energy and performance impact TrustZone-based TEEs and frameworks have on secure services. Understanding the limitations of TEEs and the implication in energy and performance is crucial. Tools can assist developers when writing secure services to detect bottlenecks early in the development process. This allows developers to adjust designs of secure services during development and avoid expensive changes when secure services are already deployed in production. Such tools can also be used for configuration and calibration of secure services in order to chose appropriate sets of parameters for optimum performance. The energy consumption of IoT devices is indirectly affected by optimizing secure services for performance, which extends energy resources for longer uptime and availability of secure IoT services.

Despite existing literature on many TEE technologies and to the best of one's knowledge, an extensive study prior to this work on the energy and performance impact of TEEs has not been done. The lack of such an extensive study is addressed in this chapter by understanding the limitations of TrustZone with OP-TEE and by measuring the energy and performance impact they have when shielding IoT applications or services. IoT services rely on data shared over the Internet, which makes evaluating memory-bound operations on TrustZone particularly important. Section 3.2 presents the state of the art on network performance tools and related work on secure IoT services. The architecture of `iperfTZ`, a tool for detecting network performance bottlenecks and for configuring network parameters of secure services, is described in section 3.3. Section 3.4 describes the threat model that `iperftz` is confronted with. A first series of benchmarks in section 3.5 evaluates performance and overhead of the shared memory and secure storage implementations in OP-TEE as use cases for secure services. The final benchmark demonstrates how performance and overhead affect networking as measured during the previous benchmarks and make it a bottleneck for secure services. The discussion in section 3.6 is emphasizing several mitigation techniques for avoiding such inconveniences and summarizes the benefits of tools during development of secure services.

3.2 State of the Art and Related Work

3.2.1 ttcp

`ttcp` was developed in 1984 [282] in order to measure the performance of different TCP stack implementations for 4.3BSD. The tool is capable of measuring network bandwidth and other properties for TCP and UDP protocols. A client-server model is used by `ttcp` for measuring the bandwidth between two machines. Any protocol parameters can be configured over the command line before executing the tool. Many other network performance tools are derived from `ttcp` with additional features that were not originally provided. Among these tool are `nuttcp` and `iperf`.

3.2.2 iperf

`iperf` [235] was a network performance benchmark tool developed by Distributed Applications Support Team (DAST) of the National Laboratory for Applied Network Research (NLANR) during early 2000s. It supported the stream protocol TCP as well as the datagram protocol UDP and could adjust various protocol characteristics and parameters. After network performance measurement, the tool presents a summary on bandwidth, latency and packet loss ratio. Version 2 of `iperf` included new features, such as a bidirectional mode, in which `iperf` can be used on one end as client and on the other end as server. `iperf` was no longer maintained with the beginning of development for `iperf3` [106] by ESnet and the Lawrence Berkeley National Laboratory. `iperf3` is a complete redesign of the original `iperf` tool that aims for smaller and simpler code. Consequently, `iperf3` is incompatible with `iperf`. In the meantime new maintainers have picked up the development of `iperf` version 2 and made it available as `iperf2` [208]. Currently, `iperf2` and `iperf3` are being actively developed, yet both tools are incompatible and provide different sets of features and supported protocols.

3.2.3 Diggi

Diggi [250] is a privacy-preserving IoT, mobile and cloud system that enables any platform to securely connect and integrate at the edge. The system uses meta-code [149] to improve edge-node utilization by migrating the meta-code between computing devices at runtime. TrustZone is used in Diggi for secure remote execution of meta-code at the edge. The multi-layered architecture of Diggi spans from edge devices with TrustZone TEEs to public clouds using SGX.

The authors have evaluated the world switching overhead in TrustZone for a prototype TA. They compare the Interprocess Communication (IPC) overhead between an application running in the REE and a TA in the TEE with two processes running in the REE. The difference to the shared memory baseline evaluation in section 3.5.1 is the involvement of IPC, which does not take into account the cost incurred from isolating the security-critical code. The evaluation is done on a Raspberry Pi 3B using OP-TEE presumably v2.2.0 with 4.6 Linux kernel. From the evaluation description it is not clear which IPC has been used between the two processes running in the REE. The prototype TA is not providing any service and any invocation of the TA will immediately return. In-

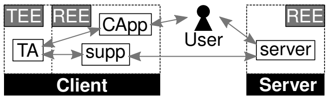


Figure 3.1: Client-server interactions between `iperfTZ` components

strumentation for time measurements have been inserted at all exception layers for a detailed breakdown of all stages involved in the communication.

According to their measurements they observe a 2.5× overhead of which single world switches contribute to about 10 % of the total communication overhead. With 60 % of the overall time the secure monitor and OP-TEE OS are using the largest amount of all stages. Their measurement of the world switching overhead in TrustZone using OP-TEE on a Raspberry Pi 3B is very much in line with the overhead from shielding a key-value store in section 3.5.1, with the exception that the baseline does not involve any IPC.

3.3 Architecture

`iperfTZ` is a secure service inspired by the network performance tools `iperf`, `iperf2` and in particular `iperf3`, which measures network performance of TAs, identifies performance bottlenecks and assists in determining appropriate network parameters. The secure services is presented in Figure 3.1 and split into the following three components.

3.3.1 `iperfTZ`: Client Application

The CApp is the primary user interface to `iperfTZ` and acts as a proxy to the TA by forwarding network parameters over shared memory. Two shared memory regions are used by the CApp to forward connection and network protocol parameters to the TA, and to retrieve network performance measurement results from the TA. The shared memory buffer for the connection and parameters contains information such as the IP of the target server, the size of the payload, and socket buffer sizes. As soon as the CApp calls the `TEEC_InvokeCommand` function to pass the parameters over to the TA and start the network performance measurement, the CApp gets blocked until the TA has completed the measurement. When the TA returns from the measurement, an SMC instruction is executed to restore the state of the CApp and to resume with the return from the `TEEC_InvokeCommand` function. Then, with the measurement results, the CApp writes the content of the shared memory to files.

3.3.2 `iperfTZ`: Trusted Application

The TA contains the core logic of the `iperfTZ` secure service. When it gets invoked, it reads the parameters from the shared memory and sets up the interface sockets for the network performance measurement. For the measurement, the TA takes the role of the client in the client-server model. Before starting the measurement payloads are dynamically allocated and filled with randomly generated data from the Cryptographic Operations API [120]. Time is measured using the Time API [120], which is implemented in OP-TEE using the physical count and the frequency registers. The TA has three modes, in which it can either send packets at constant rates, send specific amounts of bytes or run for exactly 10 seconds. During the measurement information is gathered about the number of transmit calls, *i.e.*, `recv` and `send`, the number of bytes sent, the time spent in transmit calls, and the total runtime. These measurement results are then written to the share memory area when the measurement has completed and before the TA initiates a world switch back to the CApp.

3.3.3 `iperfTZ`: Server

The server is a regular application executing in a REE and form the counterpart to the TA. The server has to be started before and with the same parameters as the CApp for a proper setup of the network performance measurement. After setting up the sockets according to the parameters, the server is waiting for incoming TCP connection or inbound UDP datagrams from the TA. During the measurement the sever collects the same information as the TA. Additional information is retrieved from the Linux kernel for TCP measurement, such as the smoothed round trip time or the maximum segment size. This information is not accessible to TAs and is retrieved from the server using a `getsockopt` system call in order to complement the measurement results.

3.4 Threat Model

Malicious users can gain physical or remote access to devices running `iperfTZ`. By compromising the devices or the network devices are connected to, malicious users can control the devices and use them for DoS attacks. The TCB of `iperfTZ` includes the firmware (*i.e.*, dispatcher and secure monitor), the trusted OS OP-TEE and the `iperfTZ` TA. Everything else is not considered to be part of the TCB and cannot be trusted. The use of eMMC improves the availability of the secure service by preventing malicious users from denial of service attacks through disabling TAs from being loaded from the REE file system. During network performance measurements malicious users can run (distributed) denial of service attacks to reduce available network bandwidth and increase network latency, such that `iperfTZ` measures and reports incorrect network performances. `iperfTZ` does not provide support for the TLS protocol because OP-TEE did not provide an implementation at the time. Although irrelevant to `iperfTZ`, this means that any traffic between two `iperfTZ` instances can be intercepted by using man-in-the-middle attacks. The network traffic can be intercepted either in the REE when the network stack is borrowed from the rich OS or on the network.

3.5 Evaluation

The evaluation was carried out using OP-TEE on a Gibloux devices and on a VM running on Pilatus while emulating as closely as possible a Raspberry Pi platform. It is for this reason that the processor of the default QEMU setup in OP-TEE was changed from an ARM Virtual Machine `virt` with two cores to an ARM Cortex-A53 with four cores. QEMU is used in system emulation mode, where it emulates an entire machine and dynamically translates the ISA of the

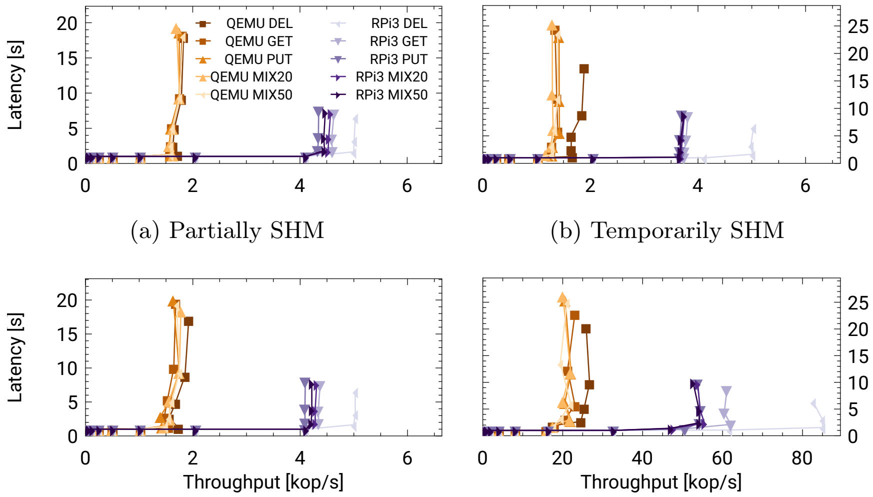
VM to the host ISA. The default SLiRP network [258], the VM is connected to, is replaced by a bridged network with a tap device. In particular the Gigabit network bandwidth on Pilatus is important for the evaluation of `iperfTZ`, since on Gibloux network bandwidth is limited to Fast Ethernet at 100 Mibit/s.[1] Pilatus is running Ubuntu 18.04.1 LTS on a 4.15.0-43-generic Linux kernel. For the evaluation of the shared memory and the secure storage OP-TEE v3.2.0 and QEMU v2.12.0 were used. For the evaluation of `iperfTZ` OP-TEE v3.5.0 and QEMU v3.1.0-rc3 were used. The `iperfTZ` server is running for the evaluation on Gibloux and Pilatus on the same auxiliary server and has a Gigabit Ethernet connection. All devices are connected through Gigabit Ethernet to the same network switch. The power consumption of Pilatus is recorded using the PDU, while the power consumption of Gibloux is recorded using the PowerSpy.

3.5.1 Shared Memory

A simple key-value store TA was implemented, in order to evaluate the overhead and performance of data sharing between an application running in the REE and in the TEE. The key-value store is a modified version of the hash table in *kazlib* [174] v1.20 for which support of multiple context and dynamic tables have been removed. The resulting hash table is a static hash table with 251 chains that uses modular hashing and separate chaining for collision resolution. The modified static hash table is used as baseline in the REE and has been ported to a key-value store TA for evaluation in the TEE. Overhead and performance are measured by the application running in the REE. Monotonic wall-clock is used as time source for benchmarking `DEL`, `GET` and `PUT` operations. The `DEL` operation is the least memory-intensive operation, which looks up a data object in the hash table and frees its memory. The `GET` operation retrieves a data object from the hash table and copies it into shared memory, which results in a lower overhead. The `PUT` operation is the most memory-intensive operation with high overhead, requiring memory allocation in the TEE, copying data objects from shared memory into allocated memory and inserting data objects into the hash table. Two mixes of `GET` and `PUT` operations are also evaluated, wherein the number in the name of the mix indicates the percentage of `PUT` operations. `MIX50` is more memory-intensive than `MIX20` due to the additional overhead from the 30 % of `PUT` operations. All operations are uniformly distributed and invoked 256 times at rates between 1 to 32 768 operations per second.

The overhead and performance of three types of shared memory are evaluated. These are whole, partially and temporarily shared memory. Each shared memory consists of a memory block of 512 KiB that is filled with random data from `/dev/urandom`. This random looking data mimics the use of encrypted data that is transferred from the application in the REE to the key-value store TA for safekeeping. A random offset into the memory block is computed before every invocation to determine the 1 KiB data object that is to be used for the next key-value store operation. The benchmark for the `DEL` and `GET` operations begins a pre-populated hash table of 256 data objects, while the benchmark for the `PUT` operation begins an empty hash table. For the benchmark with the operation mix of `GET` and `PUT` operations, the hash table is pre-populated relative

[1]It should be noticed that the Raspberry Pi 3B+, which was supported later on by OP-TEE, uses Gigabit Ethernet, but network bandwidth is limited at 480 Mibit/s by the USB 2.0 bus interface.



(a) Partially SHM

(b) Temporarily SHM

(c) Whole SHM

(d) Inside REE

Figure 3.2: Throughput-latency plots of shared memory types for key-value TA in TEE and REE

to the percentage of `GET` operations.

The results of the shared memory evaluation are shown as latency and throughput in Figure 3.2. In general memory-intensive operations have lower throughput than less memory-intensive operations. For this reason the following throughput order from lowest to highest for the key-value store operations is expected: `PUT`, `MIX50`, `MIX20`, `GET`, and `DEL`. For the baseline in Figure 3.2-d, the operations reach distinct maximum throughput depending on the type of operations. Only throughput for memory-intensive operations `PUT`, `MIX50` and `MIX20` is overlapping in the baseline. Comparing the baseline measurement to the shared memory measurements, it should be noticed that shielding the key-value store in the TEE incurs an overhead of more than one order of magnitude. On Pilatus the overhead is between $12\times$ to $14\times$, and on Gibloux the overhead is between $12\times$ to $17\times$. The overhead is consistent between emulation with QEMU and the Raspberry Pi platform. Partially and whole shared memory achieve very similar throughput, since their memory regions are mapped in advance to the TA. The memory region of temporarily shared memory has to be mapped to the TA at runtime which incurs an overhead of about 15 % for the evaluation on Gibloux. With QEMU it is difficult to determine any overhead since TrustZone and other components have to be emulated and operations cannot be clearly distinguished, despite being of different types.

3.5.2 Secure Storage

Performance and overhead of the secure storage implementation in OP-TEE are evaluated using the secure storage benchmark from the OP-TEE sanity test suite [189]. `WRITE`, `READ` and `REWRITE` commands are evaluated in the benchmark for data objects with a size between 256 B to 1 MiB. Data objects are accessed in chunks of up to 1 KiB upon execution of secure storage commands. Neither

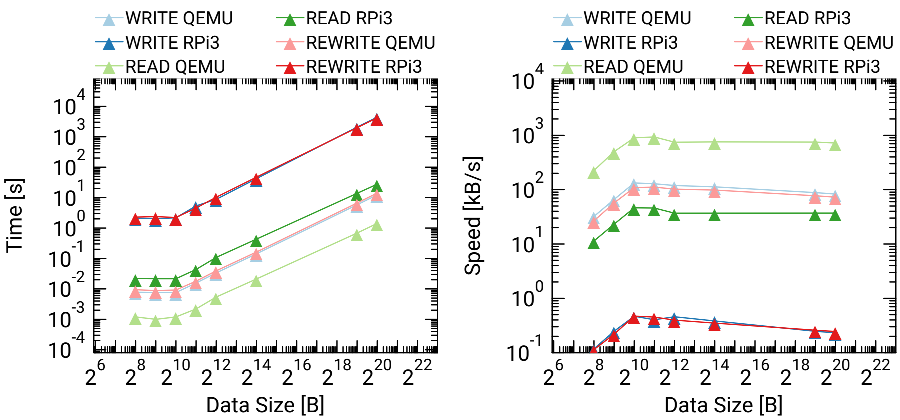


Figure 3.3: Secure storage benchmark execution time and throughput

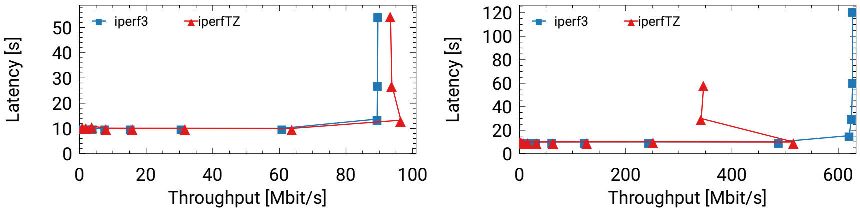
Gibloux nor the QEMU environment provide an eMMC for secure storage. For this reason the evaluation uses the secure storage implementation of OP-TEE that is provided from the REE.

The command execution time and speed (*i.e.*, throughput) recorded during the secure storage benchmark are shown for various data chunk sizes in Figure 3.3. The figure to the left indicates how the command execution time is increasing when data objects larger than the data chunk size of 1 KiB are accessed. The linear part of the curve indicates an increase in execution time for commands accessing data objects larger than data chunk sizes due to multiple accesses. Similar to the previous shared memory evaluation in section 3.5.1, memory-intensive commands such as `WRITE` and `REWRITE` have longer execution times than the less memory-intensive command `READ`. The results are consistent across both evaluation platforms, wherein QEMU has shorter execution times due to higher CPU frequency and wider memory as well as storage bandwidth than is available on Gibloux.

The results for the speed of the secure storage to the right of Figure 3.3 come to the same conclusion as for the command execution time. The highest speed is achieved when the data object sizes are equal to the data chunk size of 1 KiB. Speed is increasing until reaching the data chunk size. Beyond the data chunk size the speed remains constant with a slightly negative tendency. The constant speed is explained by multiple accesses to the secure storage as a consequence of the smaller data chunk size.

3.5.3 Interface Sockets

This evaluation measures the network performance using `iperfTZ` on interface sockets implemented in OP-TEE and compares it the network performance measured using `iperf3`. `iperfTZ` and `iperf3` use the same configuration setup of 128 KiB socket buffer and packet payload sizes. The maximum sizes of read and write socket buffers in the Linux kernel is obtainable through the procfs pseudo file system interfaces `/proc/sys/net/core/rmem_max` and `/proc/sys/net/core/wmem_max`. The socket buffer limits of the Linux kernel can be adjusted at runtime through the previously mentioned interfaces. Due



(a) Raspberry Pi

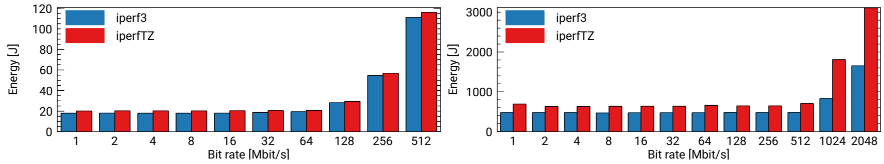
(b) QEMU

Figure 3.4: Throughput during TCP network performance measurements.

to the TA being limited to 1 MiB of memory on the Raspberry Pi platform, not more than 512 KiB are allocated on the heap of the TA for generating packet payloads. A series of constant bit rate measurements is taken, such that after each measurement the bit rate is doubled starting at 1 Mbit/s up to 4 Gbit/s.

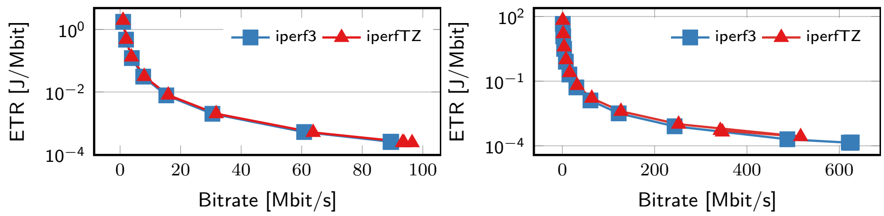
The results of the network performance measurements while maintaining constant bit rates are shown in Figure 3.4. The throughput of `iperfTZ` exceeds in general the throughput of `iperf3` on both platforms by up to 9 %. This can be explained by the simplified main loop in `iperfTZ` compared to `iperf3`. The TA of `iperfTZ` cannot provide the same features in the TEE as `iperf3` in the REE, and has fewer condition to validate. On Gibloux the network throughput reaches saturation close to the Fast Ethernet limit. However, on Pilatus network throughput of `iperfTZ` is degrading before reaching the saturation point of `iperf3`. High throughput rates beyond 512 Mbit/s appear to be more affected and degrade beyond unaffected throughput rates. This is a clear indication that world switching is interfering with the measurement. The memory throughput observed in section 3.5.1 is close to 100 MiB/s, while the Fast Ethernet network throughput is limited to 10 MiB/s. The throughput gap between the two evaluations amounts to almost one order of magnitude, which appears to be sufficient for the world switching overhead not to interfere with the network throughput on Gibloux. Given that the shared memory evaluation of section 3.5.1 is comparable to this interface socket evaluation, one would expect to observe saturation due to the world switching overhead around 4 MiB/s, respectively 32 Mbit/s, on Gibloux. This is not the case and most likely due to buffering several packets compared to single key-value store operations. The world switching overhead affects network throughput on Pilatus in the range of 512 Mbit/s to 650 Mbit/s. By comparing the ratio of the memory throughput to the network throughput on Pilatus, it seems that interface socket buffers have at least 32 × the size of the key-value payload size. The world switching overhead affects the network throughput of interface sockets by a factor of 1.8 ×.

The energy consumed during TCP network performance measurements is shown in Figure 3.5. Each bar in the plot indicates the total amount of energy that was consumed by the devices when measuring the network performance at a constant bit rate. `iperfTZ` is consuming on Gibloux about 2 J, *i.e.*, 11 %, more energy in the unsaturated region than `iperf3`. Then energy is doubling along with bit rates in the saturated region between 128 Mbit/s to 512 Mbit/s. Similar results can also be observed on Pilatus. On Pilatus the energy difference is larger, since measurements are done on a server-grade platform and because `iperfTZ` is operated inside a VM executing in system emulation mode.

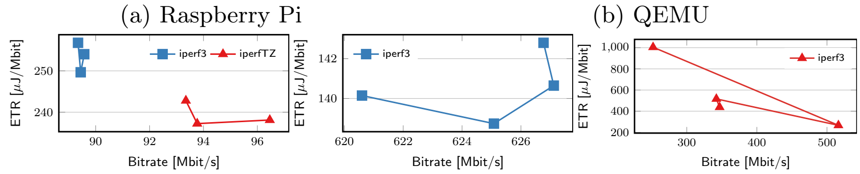


(a) Raspberry Pi (b) QEMU

Figure 3.5: Energy consumption during TCP network performance measurements. Bit rates on the x-axis are given as binary logarithm.



(a) Raspberry Pi (b) QEMU



(c) Raspberry Pi (d) QEMU iperf3 (e) QEMU iperfTZ

Figure 3.6: Energy efficiency during TCP network performance measurements. ETR on the y-axis of figures (a) and (b) are given as decimal logarithm. Figures (c) to (e) are magnifications of the saturated regions in figures (a) and (b).

`iperfTZ` is consuming on Pilatus about 173 J, *i.e.*, 36 %, more in the unsaturated region than `iperf3`. In the saturated region between 512 Mbit/s to 2048 Mbit/s energy consumption for `iperf3` is almost doubling, while energy consumption for `iperfTZ` is more than doubling. The difference in energy consumption between `iperfTZ` and `iperf3` on Pilatus can be attributed to the world switching overhead. Energy consumption is affected by a factor of $1.6\times$ from the world switching overhead.

Figure 3.6 is showing the energy efficiency during the TCP network performance measurements. The energy efficiency for `iperf3` and `iperfTZ` are decreasing on the Raspberry Pi and with QEMU. This is a clear indication for amortization, because the systems are not running at maximum throughput. Idle time, that is introduced in order to match the target bit rate, is reduced as the bit rate is increased, which improves the energy efficiency. `iperfTZ` has in general worse energy efficiency than `iperf3` due to the shielded execution as shown in Figure 3.7. However, on the Raspberry Pi, the energy efficiency of `iperfTZ` gradually improves as the bit rate increases and surpasses the energy efficiency of `iperf3` in the saturated region between target bit rates 64 Mbit/s to 512 Mbit/s by an average of 5 %. In the saturated region optimum energy efficiency points are identified as seen in Figures 3.6c to 3.6e. For `iperf3` the optimum energy efficiency is at 89.4 Mbit/s on the Raspberry Pi and at 625 Mbit/s

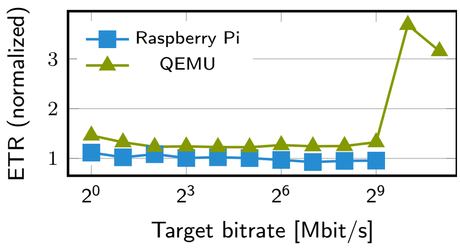


Figure 3.7: Normalized ETR comparing the energy efficiency of `iperfTZ` over `iperf3` on the Raspberry Pi and QEMU.

with QEMU. For `iperfTZ` the optimum energy efficiency is at 93.8 Mbit/s on the Raspberry Pi and at 516 Mbit/s with QEMU. The world switching overhead has a significant impact on the energy efficiency by a factor of up to $3.7\times$ as observed in Figure 3.6e and Figure 3.7 on QEMU. Overhead from emulating TrustZone with QEMU amounts to a decrease in energy efficiency of 27 % on average.

3.6 Discussion

The energy efficiency, scalability and overhead from shielding secure IoT services have been identified and evaluated in a series of benchmarks for shared memory, secure storage and network performance. It is well known that the design of TrustZone incurs an overhead for requesting services from TAs. In particular the necessity to traverse all ELs twice for every request that is made to a TA, once for taking the exception and once for returning from the exception, is introducing a significant latency. Furthermore, since the processor core needs to execute sequences of instructions synchronously in order to perform the world switch, there is no other way to speed up the process than to minimize the instruction sequences. Scalability of TAs is limited by the secure memory available to the TEE. Deploying memory demanding TAs requires reducing the number of concurrently executing instances, which lowers both the energy efficiency and the scalability. Determining the appropriate ratio between normal and secure memory is important for optimizing energy efficiency and scalability.

The results for the shared memory and secure storage evaluations clearly demonstrate these limitations of the world switch in the TrustZone design. The IPC between CApps in the REE and TA in the TEE as well as the limited memory available to TAs are the main reason for the bottleneck on memory bound secure services that require access to large confidential datasets. Service requests to TAs are unproblematic as long as these entail only a few infrequent world switches. Any requests with multiple world switches occurring frequently are lowering performance and increasing energy consumption because of the overhead. These frequent world switches result especially from network-enabled TAs, when packets are being sent or received. The impact of the world switch strongly depends on the design of the IoT service. Throughput degraded by up to one order of magnitude for the secure IoT services that have been evaluated.

Providing network functionalities for secure services requires APIs with support for sockets. However, by adding more functionalities for secure services, more code has to be included in the TCB. Two approaches are possible with Trustzone: 1) implementing drivers for direct access to NICs in the secure world kernel or 2) borrowing the network stack implementation of the REE. Developers of secure services have to trade performance and energy off against security. OP-TEE does not provide NIC drivers for OP-TEE OS, and instead opts for the secure approach of borrowing the REE network stack through the `tee-supplicant` at the cost of performance and energy. This also means that any data forwarded to the network stack of the REE has to be encrypted before leaving the TEE in order to not violate confidentiality guarantees. With secure NIC drivers additional encryption of data is not necessary, which simplifies confidential data handling.

`iperfTZ` is a network performance tool for secure services that provides more accurate measurements than `iperf3` and assists developers in detecting bottlenecks. The experimental results reveal the performance and energy tradeoff for running network-enabled secure services on hardware and in emulated environments. The world switching overhead of TrustZone decreases performance by a factor of $1.8\times$, increases energy consumption by a factor of $1.6\times$ and reduces energy efficiency by a factor of $3.7\times$. On the Raspberry Pi platform, Gigabit Ethernet cannot be used by secure services at its full capacity because the world switching overhead is the limiting factor. This problem can be mitigated by increasing buffer sizes for interface sockets and reducing the number of world switches. However, limited memory resources of TAs are restricting buffer sizes, such that the problem can only be delayed. When performance, and indirectly also energy, are important properties of secure services, then it is unavoidable to increase the TCB and provide a secure network stack implementation for the TEE. A careful preliminary assessment of secure service requirements is important in order to meet design and development demands.

An issue with the HUK is that most ARM platforms do not provide free or open access to the HUK. Without access to the HUK, key derivation in OP-TEE, for example of a storage key for the secure storage, is insecure because keys are provisioned with the software and not hardware protected. In some cases access to the HUK can be restricted by NDAs between the OEM and the service provider. Other platforms, such as the Raspberry Pi, do not provide latest documentation for their 3B and 4B SoC models, which raises the question whether HUKs actually exist on this platform. The problem of OEMs with providing access to HUKs is that the keys are located in a sensitive area of the SoC with access to other security-critical components of the SoC. By opening up access to the HUK, OEMs would publicly expose their security architecture of the SoC. In case of design flaws in the security architecture, a large number of devices could immediately be affected. While some design flaws could be patched with firmware updates, others might require hardware changes which would leave the affected devices insecure.

The following chapter 4 is presenting the development of a network-enabled secure IoT service by applying best practices for energy efficiency and scalability according to the findings of this chapter. It is explained how the service is partitioned to isolate the security-relevant component in the TEE and what other design decisions were taken to improve energy efficiency and scalability of the final IoT service.

4

Developing Secure IoT Services

4.1 Introduction

DLT is a key technology for the industry 4.0 opening up many new business opportunities. Interactions between parties can be automatically supervised by smart contracts that have been negotiated prior to their deployment. Mutually untrusted parties have to be brought together in order to participate collaboratively in the execution of smart contracts as well as the validation and order of transactions through consensus protocols. Private blockchain networks can partially resolve these trust issues by providing access control. However, attacks and compromised systems undermine the integrity of privately deployed smart contract networks. Existing smart contract networks do not provide any confidentiality guarantees and expose not only the content and data of smart contracts but also sensitive information such as assets and business interactions.

TEEs are being deployed in cloud data centers to bridge trust issues between cloud users and cloud providers as well as to encourage the deployment of large-scale privacy-preserving systems. The issues with respect to the trust relation between cloud user and cloud provider is similar to the one between mutually untrusted parties in private blockchain networks. Establishing trust between parties by using TEEs to shield the smart contract, its data and transactions is an enabling factor for such new business interactions in the industry 4.0. Through TEEs smart contracts are augmented with confidentiality guarantees to facilitate confidential smart contracts.

Regulations which protect personal data by law and restrict their use are challenges many services have to overcome. For example, machine learning services, that apply federated learning for processing images of cancer tissues distributed over several hospitals and institutions, are limited by legal regulations. Confidential smart contracts can be a solution to federated learning while complying with legal restrictions. Furthermore, confidential smart contracts are particularly interesting for supply chain and business-to-business use cases. Tracking assets transparently and processing contracts automatically according to well defined conditions are compelling arguments for using confidential smart contracts in supply chains.

In the previous chapter 3, the limitations of OP-TEE as a TEE have been identified and its impact on energy efficiency and performance was measured. Developing network-enabled secure services was found as particularly challenging because of the overhead from frequent world switches. A secure IoT service for confidential smart contracts is developed in this chapter by leveraging the previous findings on energy efficiency, scalability and security. FOC is a prototype for Fabric executing smart contracts confidentially in TEEs available on IoT devices. Although resources on IoT devices are limited, FOC demonstrates how DLTs benefit from energy-efficient architectures. TEEs provide hardware security guarantees that have orders of magnitude better performance than software-based solutions with limited impact on power consumption.

The state of the art for smart contracts is described in section 4.2 and related work is mentioned in section 4.3. Sections 4.4 to 4.5 introduce DLTs and detail the architecture of Fabric. The architecture of FOC and design improvements are outlined in section 4.6. The threat model considered for confidential smart contracts is explained in section 4.7. Energy efficiency, scalability and security of FOC are evaluated in section 4.8 for a small-scale deployment. Section 4.9 concludes this chapter with a discussion on the development of secure IoT services.

4.2 State of the Art

Ethereum

Ethereum [59] is a permissionless blockchain network that introduced the concept of executable code on the blockchain. Code is executable on the blockchain by creating a Turing-complete cryptographic ledger. This ledger is based on a stack machine executing bytecode and is known as Ethereum Virtual Machine (EVM). Ethereum relies on a cryptocurrency called Ether for encouraging network security by PoW and for paying transaction fees. The executable code is known as smart contracts that are addressable within the network and have their own balance. Smart contracts are automatically invoked when receiving transactions and can generate transactions depending on the conditions defined by the program logic. Real persons and smart contracts are the two entities in the Ethereum network that can send and receive transactions. Gas is a unit of execution similar to CPU cycles used in Ethereum to limit runtime of smart contracts and to compute transaction fees. Without gas malicious users could install infinitely looping smart contracts that execute denial of service attacks. Each bytecode operation as a fixed gas value which determines the cost of executing smart contracts. The transaction sender pays upfront in Ether for the gas needed to run the smart contract. When more gas was payed for than necessary, the sender receives the remaining Ether back. If insufficient gas was payed, then the transaction is rolled back and no Ether is returned. Solidity is the high-level language used for programming smart contracts, that gets compiled to bytecode for the EVM. Ethereum separates the state from the blockchain and keeps the latest balances according to the blockchain in a Merkle tree structure. Currently, Ethereum is transitioning from PoW to Proof of Stake (PoS) and is implementing sharding for increasing transaction throughput.

4.3 Related Work

R3 Corda

R3 Corda [136] is a permissioned enterprise blockchain that supports distributed applications called CorDapps. CorDapps extend the functionalities of its blockchain such as defining data types, protocols between nodes and smart contracts.

Corda is similar to Bitcoin by using a database of immutable states containing arbitrary data. The states in the database are outputs of transactions that get removed when they are used as inputs for future transactions. Transactions are not ordered and the Corda blockchain does not rely on PoW for consensus. Instead, each state points to a notary, which is a service signing transactions if all input states have not been consumed. One particularity is that transactions must satisfy the inputs and outputs of CorDapps with the additional restriction that all affected states must point to the same notary. Smart contracts are stored as ZIP archives in JAR format and makes use of its signing functionality to identify the author of the contract. Corda executes smart contracts on the Java Virtual Machine (JVM) by using a subset of the bytecode instructions. Each smart contract implements the `Contract` interface which exposes a verification function indicating whether transactions are valid or invalid. Unlike Ethereum, Corda uses stateless smart contracts with restricted interaction capabilities. Initially, Corda planned to support SGX and has been gradually integrated it into its blockchain. Conclave is a framework developed specifically for Corda to write confidential smart contracts running in SGX.

VOLT, Coco & CCF

Confidential Consortium Framework (CCF) [268] is an open source blockchain framework, formerly called Coco Framework [215] and based on VOLT [276], that enables development for confidential smart contracts by leveraging TEE technology. A distributed ledger is formed by integrating framework with blockchain protocols such as Ethereum, Quorum, Corda, Hyperledger Sawtooth and others. In order to provide consensus protocols with finality guarantee CCF establishes a permissioned blockchain network. Transactions transition through a classical order-execute architecture and need to be committed for achieving consensus. A quorum is installed as governance rule for ensuring the consensus. Components in CCF are not separated and the entire node is executed within the TEE to shield it from malicious attacks. This results in a larger TCB compared to other confidential smart contract implementations. CCF considers integrating a consortium concept that allows verifying the integrity of nodes, blocks and transactions at any time using receipts.

Hyperledger Fabric Private Chaincode

Hyperledger Fabric Private Chaincode (FPC) [51] is an extension to Fabric providing support for confidential smart contracts. FPC separates security relevant components from the remaining components and shields them inside SGX. Chaincodes and the ledger are the two components that are moved into enclaves in order to reduce the TCB to a minimum. Each chaincode enclave executes one confidential smart contract isolated from other chaincodes in an enclave. The chaincode enclave relies on the chaincode library for providing additional features such as state encryption, attestation and secure access to the blockchain state. The ledger enclave maintains the state of the blockchain within the TEE as integrity-specific metadata. An interface is exposed to the chaincode library for accessing the integrity-specific metadata and verifying the blockchain state. New blocks are validated and the blockchain state is modified accordingly. Thereupon, the ledger enclave generates cryptographic hashes for each key-value pair of the blockchain state to update the integrity-specific metadata. The enclave components are complemented by the enclave registry and the enclave transaction validator. The enclave registry maintains a list of all installed chaincode enclaves in the blockchain network. Remote attestation is completed by the enclave registry, which stores the attestation result on the blockchain. Entities participating in the network can verify attestations on the blockchain before invoking chaincodes. Transactions generated by chaincode enclaves are validated by the enclave transaction validator and ensures that transaction signatures were issued by the corresponding chaincode enclaves. The validator then forwards these transactions to the ledger enclave, which double checks transactions before committing them to the ledger.

Teechain

Teechain [190] is a protocol for off-chain payments in blockchain systems that exploits TEEs. TEEs are used in the blockchain network as distributed trusted third party. Within the TEEs Teechain maintains the global state of the blockchain network and sets up secure payment channels between participants. Trust is established between participants through remote attestation of Teechain instances. This guarantees that the secure payment channels are operated by genuine Teechain applications and that Teechain applications are running within TEEs. The protocol is designed in such a manner that participants do not have to share direct network connections and that Teechain will route transactions over multiple secure channels in order to form payment chains. Crash failures are problematic for Teechain, since the entire state is held within the TEE. This problem can be mitigated by regularly writing the state to persistent storage in encrypted form. Such a solution is only practicable for nodes with low payment rates but not for nodes with high payment rates. The authors propose as second solution to scale nodes with high payment rates horizontally and distribute them over multiple failure domains. Blockchain states held within TEEs are then replicated securely across these nodes. Prototype evaluation of Teechain has shown that the system has orders of magnitude higher performance than state of the art payment channels. In a transatlantic setup with high payment rate nodes Teechain achieved more than 33 000 tx/s with 100 ms latency.

Ekiden

Ekiden [73] is a system that is combining blockchains with TEEs for executing confidential smart contracts. It introduces a novel architecture that separates the consensus from the execution layer. Smart contracts in Ekiden are deterministic stateful programs. The contracts are executed within TEEs in order to provide strong confidentiality, integrity and availability guarantees. Ekiden blockchain networks distinguish between three types of nodes. Clients can invoke contracts by sending transactions to compute nodes. Compute nodes are platforms equipped with a TEE that execute smart contracts. Several compute nodes form together a key management committee that is in charge of all keys used by the TEEs. Attestations are generated by the TEE of compute nodes that are sent to consensus nodes. Consensus nodes are maintaining the ledger and store the contract states as well as attestations on the blockchain. Compute and consensus nodes operate in parallel, so that compute nodes apply speculative execution on transactions without waiting for responses of consensus nodes. The implementation of Ekiden does not support cross-contract calls, but considers including them in future work. Evaluations of Ekiden compared to Ethereum indicate $660\times$ higher throughput and $400\times$ less latency with this new architecture.

4.4 Distributed Ledger

The introduction of Bitcoin [231] at the peak of the financial crisis by Satoshi Nakamoto did not only propose a solution to the Byzantine generals problem [177] for global scale peer-to-peer public networks but it also established the blockchain as the first DLT. Few people know that the idea of distributed ledger systems was not new and had been described many year before using different terms by privacy-preserving pioneer David L. Chaum [66, 67]. Chaum describes a system of mutually untrusted parties that uses multiple vaults connected over a peer-to-peer network and public key cryptography. On closer inspection the system corresponds to a blockchain system without solution to the Byzantine generals problem[1].

The blockchain is a distributed database [308] that assembles transactions in blocks and and cryptographically links [132, 211] the blocks by the hash of the previous block in order to form a single chain. At the very beginning of every blockchain is the genesis block. Genesis blocks are particular, because they do not contain a hash to link to any previous block. The very first transaction of every block in Bitcoin is the coinbase. Coinbases are special transactions that create new Bitcoins and reward them to the miner of a new block. Bitcoin limits the total amount of coins in circulation by halving rewards for new blocks every couple of years until the rewards round to zero.

Miners are peer nodes [2] that participate in the consensus protocol over the peer-to-peer network of the blockchain system. They each assemble several transactions into a block and reach consensus among other peers solving a cryptographic puzzle in order to determine a total order on transactions. In the case of Bitcoin this puzzle consists in finding a number called a nonce, so that when the block is hashed together with the nonce, the resulting hash has a certain number of leading zeros. The peer to solve the puzzle first will disseminate the new block in the network. Other peers receiving new blocks first validate the block before appending it to their blockchain.

The underlying data model of Bitcoin and many other blockchain systems is the Unspent Transaction Output (UTXO). The UTXO is a database formed by a set of immutable rows containing hashes of transaction outputs that have not yet been spent. Every transaction, except coinbases, can have multiple inputs and multiple outputs in Bitcoin. For any new transaction, the inputs would consume the corresponding rows in the UTXO and the outputs would add new rows to the UTXO. Besides blockchain as fundamental technology there are also Directed Acyclic Graph (DAG) [38, 252] DLTs. However, due to the lack of independent consensus evaluations for DAG-based DLTs many open questions remain [60].

[1] The Byzantine generals problem was published during the PhD thesis of Chaum.

[2] From here on referred to as *peers*

4.4.1 Control Policies

The access to blockchain systems is regulated by control policies. Anyone can participate in a *public* or *permissionless* blockchain without the need for an identity. Examples of permissionless blockchain systems are Bitcoin and Ethereum. Their security model is based on consensus protocols relying on computational power from CPUs, GPUs and Application-Specific Integrated Circuits (ASICs) as well as economic incentives. In a *private* or *permissioned* blockchain system only known entities are allowed to participate. These entities share a common goal but they might mistrust some of the other participants. A *consortium* blockchain is a hybrid of permissionless and permissioned blockchains. As with permissioned blockchains, only known entities are allowed to participate in a consortium blockchain. However, like in permissionless blockchains, there is not a single central entity in control of the blockchain. Although entities do not necessarily trust each other in consortium blockchains, they need to agree on the service to deploy, which includes the code, its governance and the initial configuration.

4.4.2 Consensus Protocols

Proof of Work

PoW is a consensus protocol that is based on the principle of solving cryptographic puzzles and has been introduced together with the Bitcoin [231]. Peers receive transactions that they pre-execute in order to verify their validity. Several valid transactions are then assembled to a block by each peer. The assembly of a block is the pre-condition before peers can proceed with solving the cryptographic puzzle. When peers solve the puzzle, they disseminate the block over gossip protocols to all other peers in the blockchain network. Upon reception of a new block, peers verify the validity of all transactions in the block and the solution to the puzzle. If the transactions in the block and the solution to the puzzle are valid, then peers execute the transactions sequentially.

Proof of Stake

PoS [168] is an energy independent alternative security model to PoW. It is based on the concept of coin age which is the amount of currency multiplied by the holding period. Transactions need an additional timestamp field in order to determine the holding period. Like Bitcoin, PoS relies on a special transaction called coinstake. Coinstakes have several inputs with the kernel being the first input and stakes being the remaining inputs. The kernel indicates the hash target that has to be met over a limited search space, meaning that hashes of transaction assembled to blocks must match the kernel. When a new block is created the validator is rewarded but its coin age of the input stake is also reset. This mechanism ensures that peers cannot repeatedly be selected as validator for blocks. Coins are minted with respect to the consumed coin age in the coinstake transactions. In case of a fork in the blockchain the chain with the higher coin age wins.

Byzantine Fault Tolerance

Byzantine Fault Tolerance (BFT) protocols permit a certain number of Byzantine nodes which have been compromised in order to work against other nodes when trying to come to an agreement. These protocols are based on State Machine Replication (SMR) [176, 273]. SMR allows replicating services performing deterministic operations in a distributed system. In blockchain systems all peers have to execute transactions in the same order which can be achieved by SMR or more generally using BFT protocols. Among permissioned blockchain systems Practical Byzantine Fault Tolerance (PBFT) [61] and derived protocols are widely used. With PBFT safety and liveness properties [175] can be provided with up to one third of faulty peers.

4.4.3 Order-Execute Architecture

Order-Execute [25] is a typical architecture many blockchain systems are using such as for example Bitcoin or Ethereum. The architecture is describing the two phases in the system transactions are going through. In many blockchain systems consensus protocols are used to order transactions first before executing them in the previously determined sequence.

Scalability of this architecture is limited by the sequential execution of transactions. For smart contract blockchain systems throughput can also be influenced by smart contracts that are deployed by users. Malicious users could deploy infinitely looping smart contracts as DoS attack which due to the halting problem [77, 292] cannot be detected. Blockchain systems with cryptocurrencies avoid the halting problem by introducing limited resources users have to pay for when executing smart contracts. The requirement of a cryptocurrency makes this approach impracticable for blockchain systems not relying on cryptocurrencies. Parallelization of transaction execution would improve scalability but is limited by several challenges [25].

Smart contracts need to be deterministic in order to avoid inconsistencies among peers on the state of the ledger. Users could be required to write deterministic smart contracts in general purpose programming languages, or blockchain systems could provide Domain-Specific Languages (DSLs) that would only allow deterministic operations. Many general purpose programming languages make use of non-deterministic constructs users might not necessarily be aware of. Smart contracts would have to be scanned for such non-deterministic programming language constructs. The DSL approach has the disadvantage that users need to learn an additional programming language.

4.5 Hyperledger Fabric

Fabric [25] is an open source project that implements a permissioned blockchain with support for smart contracts. Unlike permissionless blockchains Fabric does not depend on computationally intensive and resource demanding consensus protocols such as PoW or PoS. Instead Fabric uses interchangeable BFT consensus protocols. The blockchain system does not rely on a cryptocurrency and therefore smart contracts can be written in several general purpose languages. Access to the permissioned blockchain system is provided by the membership service provider. Furthermore, the membership service provider is responsible for node credentials, authenticating transactions, verifying the integrity of transactions and signing as well as validating endorsements [25]. The components of Fabric are executed in Docker containers, which allows for native deployment in the cloud. The architecture of Fabric deviates from the traditional order-execute architecture of common blockchain implementations. Rather than ordering transaction inputs as order-execute architectures do, Fabric orders transaction outputs and state dependencies. Fabric uses a hybrid approach for the state replication of the ledger by passively replicating state updates before consensus and actively replicating state changes after consensus.

4.5.1 Architecture

Fabric is considered as distributed OS executing distributed applications for permissioned blockchain systems. An execute-order-validate architecture is being used instead of a classical order-execute architecture. Clients, peers and ordering nodes are the three types of nodes composing Fabric networks. Multiple blockchain systems can run in parallel on any subset of nodes and are isolated from each other through channels. Transactions are totally ordered within channels but not across channels. Distributed applications are called chaincodes which express the logic of smart contracts and are executed in separate Docker containers for isolation. The blockchain system and its configuration are modifiable through system chaincodes which are distinct from regular chaincodes. Fabric supports non-deterministic chaincodes by exchanging the execution with the ordering phase in the architecture. Chaincodes can be written in any general purpose language such that non-determinism would only affect the chaincodes themselves [25]. The ledger can only be changed by appending states resulting from chaincode invocations. Interactions between nodes in each of the phases of the execute-order-validate architecture are described in the following paragraphs.

Execution Phase

Clients send, during the first phase, transaction proposals to a set of peers according to the endorsement policy. Endorsement policies are defined by the Fabric administrator and determine which peers are invoking the chaincode. This set of peers is also known as endorsers. The chaincode invocation is then simulated in a Docker container by each endorser. Endorsers record state changes in a local key-value store that is associated to the chaincode. At the end changes are split into a *readset* and a *writeset*. The readset is holding all accessed keys of the local key-value store while the writeset is holding all modified and new values. Endorsers send signed transaction responses with the chaincode execution result, including the readset and writeset, back to clients. As soon as clients satisfy the endorsement policy by collecting enough transaction responses from endorsers, they prepare a transaction with the endorsement results for the ordering service.

Ordering Phase

During the second phase transactions are assembled from clients by the ordering service. The ordering service is a set of orderer nodes that use a consensus protocol in order to agree on a total order for transactions. For each channel the ordering services establish the total order on the received transactions. Several ordered transactions are then formed to a block and disseminated to all peers. Not only ordered are the transactions within blocks, but also the blocks on each channel are totally ordered. The ordering services provides several safety and liveness properties that can be adapted to system requirements.

Validation Phase

In the last phase peers verify that the endorsement policy of the transactions in the blocks is satisfied and that state changes do not introduce read-write conflicts. The verification of the endorsement policy and for read-write conflicts is done by the validation system chaincode. If at any stage the verification of a transaction fails, then the transaction is masked and its state update is not applied. Blocks are appended to the ledger with a mask indicating invalid transactions and the changes of valid transactions applied to the world state. The world state is essentially a cache of the latest key-value state according to the blockchain in the ledger. Invalid transactions are allowed in Fabric for example to identify potentially malicious attacks on the blockchain system.

4.6 Architecture

The architecture of FOC in Figure 4.1 has been designed and implemented in [225, 226] and is largely inspired by FPC. FOC is accounting for the limited available memory and storage resources of TrustZone-enabled devices in its modular design. The system requires the Go programming language, Fabric, OP-TEE, gRPC and their dependencies. Not only do these packages overall amount to several hundred megabytes of storage, but also would FOC need a few hundred megabytes of memory at runtime. In particular embedded and microcontroller devices cannot satisfy these requirements. With a modular design approach decomposition into smaller components allows to tailor the system to the resources of the devices.

Security-relevant components, such as OP-TEE, chaincodes and gRPC, are isolated from other components in the system. Two smaller components are introduced in order to facilitate the isolation of the security-relevant components into TrustZone-enabled nodes and to implement the modular design. The wrapper executes on the peer and is an interface between peer, ledger and chaincodes. It is responsible for installing and instantiating chaincodes on the peer and forwards any transactions to the proxy executing on a TrustZone-enabled node. Transaction responses from chaincodes are relayed to the peer. The proxy resides in the REE of the TrustZone-enabled node and communicates with the wrapper on the peer over gRPC. Any transactions received by the proxy are forwarded to the chaincodes executing in the TEE and transaction responses are sent to the wrapper. When chaincodes need to consult or change the state of the ledger, they submit those requests to the proxy and the proxy sends those requests over the wrapper to the ledger.

The ledger is maintaining in a Fabric network the block store and a peer transaction manager. Validated blocks are being written to a set of append-only files and stored locally on the node by the block store. The latest state

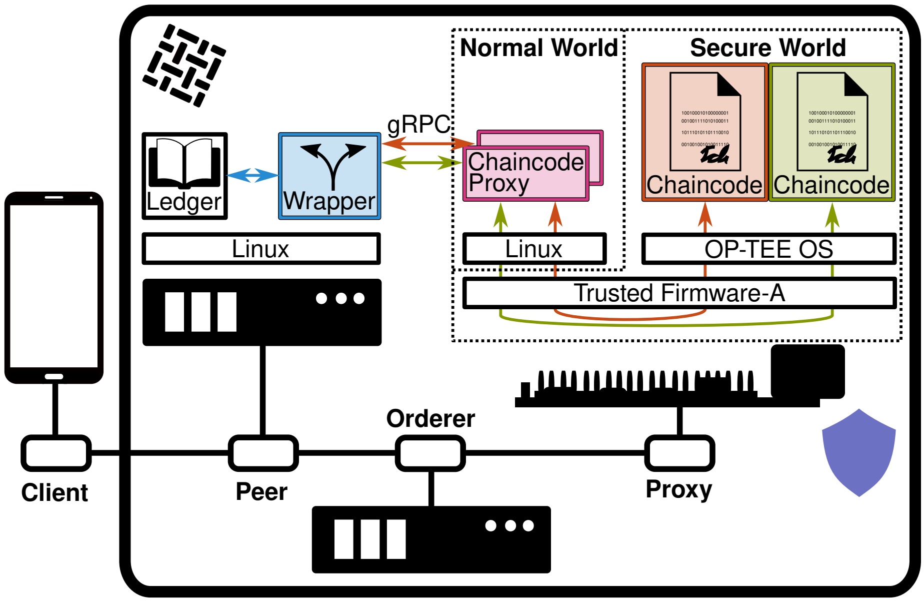


Figure 4.1: FOC architecture. Colored components indicate additions made to the original Fabric framework.

of the blockchain is maintained in a key-value store by the peer transaction manager. There are several open challenges which are currently preventing to isolate the ledger in an OP-TEE TA.

The block store requires access to a file API in order to append validated blocks to files. A standardized file API has not been defined for TrustZone-based TEEs and would result a significant increase of the TCB for OP-TEE. Alternatively, the secure storage could be used. However, the replay-protected secure storage on the eMMC is limited and inadequate for the block store. The secure storage of the REE is not storage limited but does not have the same security guarantees as on the eMMC and is restricted by the world switching overhead as described in section 3.5.2. Whether the secure storage in the REE can satisfy the requirements of the block store remain an open question.

The peer transaction manager cannot easily be shielded in a TA because either LevelDB [116] or Apache CouchDB [27] are being used by Fabric for the key-value store. LevelDB is written in Go and Apache CouchDB in Erlang, but neither of the runtime environments are being supported within OP-TEE. A custom solution written in C as in section 3.5.1 would be required. However, the limiting factor for the key value-store would be the small amount of memory available to TAs and OP-TEE. Increasing the memory of TAs to required levels contradicts the design principles of OP-TEE and cannot easily be achieved.

For these reasons, the ledger has not been shielded in FOC and the open challenges are to be addressed in future work. The ledger has access to sensitive data and could be shielded by a different TEE such as SGX. Securing the access to sensitive data by the ledger can be achieved with a hybrid-TEE solution that includes FPC. FPC could be used with FOC by extending the interface of the wrapper to the enclave registry and the enclave transaction validator. Such a hybrid solution was not considered because FOC focuses primarily on IoT devices by which the ledger is not explicitly shielded.

Several chaincodes can be instantiated simultaneously on the peer and execute inside the TEE of the TrustZone-enabled node isolated from the REE. An important difference to FPC is that chaincodes executing as TAs inside OP-TEE are not isolated from each other compared to chaincode enclaves. Chaincode replication and attestation are not considered in the initial architecture of FOC. Chaincodes are executed as TAs which requires compiling the chaincodes to native binaries. TA binaries might be incompatible depending on the ARM ISA supported by TrustZone-enabled nodes. In this situation source code of chaincodes has to be shared among all participating parties in the blockchain network and be compiled to native binaries. TA binaries would then have to be transferred to the signing authority of the blockchain network that is in possession of the private build key. The signing authority would then sign the TA with the private key and send it back to the requesting party for installation. Attestation of chaincodes with this approach is challenging, since binaries of the same source code version differ due to different ISA versions. Invoked chaincodes can be deployed and executed non-deterministically on one of the TrustZone-enabled nodes. Clients would have to account for several different attestation reports when invoking chaincodes.

VMs executing chaincodes as bytecode would overcome the previously described challenges. Sharing source code among all participating parties and building TAs for all deployed ISAs would no longer be necessary. The chaincode replication could be fully automated within the blockchain network. The same applies to DSLs for writing smart contracts that execute within VMs such as Solidity executed by EVMs. VMs have the disadvantage that they need to be shielded, because they execute the smart contract and have access to confidential and sensitive data. The TCB has to include the VM resulting in an unavoidable increase of the code size within the TEE. The additional overhead from running VMs in the TEE further limits resource availability on IoT devices.

4.7 Threat Model

FOC assumes an IoT-like deployment with servers running in enterprise or cloud data centers, while TrustZone-enabled nodes are deployed as embedded, mobile or IoT devices outside of data centers. Malicious attackers with physical access to TrustZone-enabled nodes executing chaincodes are the primary threat. The threat model does not consider attacks on the ledger by malicious administrators because the ledger is not explicitly shielded by TEEs. The TCB of FOC includes firmware, such as the dispatcher and the secure monitor, OP-TEE and chaincodes which are all shielded by TrustZone. Side-channel attacks on TrustZone [259, 269] are not within the scope of the threat model and considered orthogonal as in [29]. Mitigation techniques [185] against cache speculation side-channel attacks are continuously being updated by ARM [186]. All other components, with the exception of the ledger, belong to the REE and are considered untrusted. Components in the REE can be compromised by malicious attackers through bugs and exploits in various programs and libraries. Byzantine failures of nodes in the blockchain network can be countered by using PBFT [61] as consensus protocol.

TAs are one attack vector for compromising the TEE. Chaincodes are executed as TAs within the TEE and stored securely in the REE. At runtime TAs are not integrity protected such that bugs, exploits and other TAs could modify chaincodes. Installation or modification of chaincodes requires access to the private build key used for singing TAs. The private build key provides integrity protection and has to be kept confidential because when it leaks the entire system is compromised. Confidentiality of chaincodes can be provided by additional encryption of the TAs. Even without support for remote attestation, clients can assume that the invoked chaincode can be trusted. Fides [257] is a scheme that could provide remote attestation for FOC. Attackers would have to inject malicious code into TAs before the signing process. Source code can be subjected to additional restrictions before signing it with the private build key such as trusted development sources by signing commits and code audits.

4.8 Evaluation

The evaluation of FOC is using OP-TEE v3.8 on *Gibloux* and for QEMU running in full system emulation mode. A set of 27 *Kaiseregg* servers are used for operating the FOC network. The orderer for the underlying Fabric network is deployed on one *Kaiseregg* server, 8 peers are running wrappers, 8 nodes are emulating the TrustZone-enabled devices with QEMU and 8 nodes are used for clients invoking chaincodes. For the deployment on TrustZone hardware 8 *Gibloux* devices are used. *Gibloux* devices are operated with the Linux CPUFreq governor set to powersave for consistent power consumption. The two remaining *Kaiseregg* servers are used for recording power consumption of a subset of nodes in the FOC network. The LINDY iPower Control 2x6M PDU has a limited number of ports of which one is assigned to the orderer, three to peers for measuring the wrapper and three to nodes emulating TrustZone-enabled devices with QEMU. All *Gibloux* devices are equipped with a Raspberry Pi PoE-HAT using 802.3.af PoE. A Ubiquity Networks UniFi USW-48P-750 switch connects *Gibloux* devices to the *Kaiseregg* servers in the FOC network and provides an interface for querying power consumption. Every node in the FOC network is connected over $1\,\mathrm{Gbit\,s^{-1}}$ Ethernet links. QEMU emulates an ARM Cortex-A53 CPU with Symmetric Multiprocessing (SMP) set to 4 cores in order to match the hardware specifications of *Gibloux*. OP-TEE is configured to allow for up to 4 trusted threads for maximum scalability inside the secure world. The *Kaiseregg* servers are running Ubuntu 18.04 LTS except for client nodes which are running Ubuntu 20.04 LTS. Clocks of the auxiliary and client *Kaiseregg* servers are synchronized using NTP, in order to accurately map the evaluation measurements to the power measurement.

The chaincode implements a simple example of a coffee tracking smart contract. Coffee consumption of fictitious persons is tracked by client devices such as badges, mobile or IoT devices that are simulated by 8 *Kaiseregg* servers. Whenever persons buy or take coffee from vending machines an application on the client device will invoke the chaincode that increases the coffee count of the person by one. The application allows persons to monitor their consumption and retrieve their statistics. The simple design of the chaincode is comparable to a read/write benchmark on the FOC blockchain. Clients continuously issue transactions that are send to peers before being forwarded to the chaincode in the secure world of the TrustZone-enabled devices. Measurement of the client execution time starts before issuing the first transactions and ends after receiving the response from the last transaction. Each client instance counts the number of issued transactions during the measurement.

The chaincode consists of about 400 lines of trusted ISO C code that interacts with the proxy implemented in about 22 000 lines of untrusted C++ code mostly generated by gRPC v1.28.1. Go v1.4.1 is used for implementing the peer including the wrapper summing up to about 80 000 lines of code. Considering only platform-specific source code, OP-TEE OS adds 233 600 lines of trusted C and assembly code which together with the 31 400 lines of trusted C and assembly code from the Trusted Firmware-A complete the TCB. Separating the chaincode execution increases the TCB only by 0.2 % instead of executing the entire peer inside the secure world and contributing 23 % to the TCB. Put differently, more than 100 chaincodes could be installed before the TCB is equalizing to the TCB running the entire peer in the secure world.

4.8.1 Throughput and Latency Impact

This subsection has been included from [225] and is needed in order to explain a programming flaw in the implementation of FOC that prevents vertically scaling the number of chaincodes on TrustZone-enabled devices.

Up to eight clients are deployed to evaluate the impact of shielding chaincodes on latency and throughput. For each measurement clients consecutively invoke transactions during 30 seconds. The expected behavior of this evaluation is that latency and throughput should scale until the number of clients is equal to the number of CPU cores. From this point on latency should increase and throughput should remain stagnant. Preliminary evaluations have shown that throughput is only scaling by a factor of $1.2\times$ when doubling the number of clients from 1 to 2 clients. Two different setups were considered for pinning down the issue with the throughput overhead. The baseline setup runs all components of FOC within the REE of the same machine. The second setup is almost identical with the first setup but is not using gRPC.

The impact of gRPC on the system is evaluated by comparing the baseline setup to the second setup. Completion queues notify instances with gRPC about asynchronous events. In order to determine the ideal set of parameters, measurements are run on QEMU and the number of emulated CPU cores is set to 8. Optimum results are found by setting the number of completion queues to half the number of CPU cores without changing the default number of polling threads. Under this configuration throughput increased by $1.9\times$ for 2, $2.5\times$ for 4 and by $2.8\times$ for 8 clients by using gRPC. In this setup higher throughput is achieved compared to the preliminary evaluation of FOC. Without gRPC component the second setup achieves lower throughput due to lack of multithreading support limiting scalability of the FOC architecture between wrapper and proxy. Throughput scales by factors of $1.3\times$ for 2, $1.4\times$ for 4 and $1.5\times$ for 8 clients. The baseline and second setup have made obvious that gRPC is not the source of the overhead and that the overhead must originate from the proxy or the chaincode.

Figure 4.2 presents throughput and latency for the two setups on Raspberry Pi 3B and QEMU platforms. Both platforms use the same optimum gRPC parameters on 4 CPU cores. On Raspberry Pi 3B throughput for the baseline

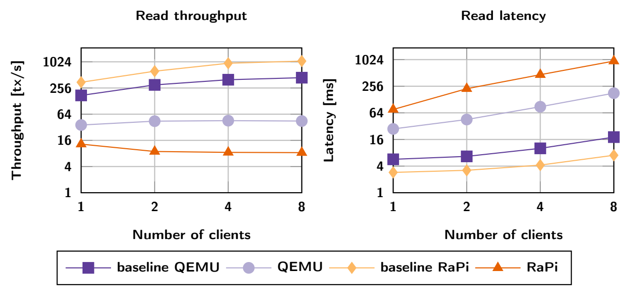


Figure 4.2: Read throughput and read latency of the coffee tracking chaincode for increasing numbers of clients.

setup scales up to a factor of 3.1 × for 8 clients. With emulation by QEMU the baseline setup throughput scales a little lower to a factor of 2.6 for 8 clients. The throughput with FOC scales up to 1.2 × for 2 clients and then stagnates up to 8 clients. Comparing FOC to the baseline setup on the Raspberry Pi 3B results in a 27 × higher throughput for a single client and a 130 × higher throughput with 8 clients. On QEMU differences are not quite as high for throughput ranging between 5 × to 10 × for 1 up to 8 clients.

In [225] the evaluation concludes that shielding chaincodes inside the secure world with TrustZone introduces a significant overhead in terms of throughput. This conclusion is also confirmed by the latency breakdown in [225] which identifies TA context and session handling functions in OP-TEE to be by far the most time consuming functions during chaincode invocations. However, an oversight in the implementation of FOC prevents sessions from being reused. This means that for every chaincode invocation a new session to the TA has to be established. During this process chaincodes are loaded from the normal world into the secure world and instantiated. On QEMU session establishment accounts for 61 % of the chaincode invocation time and session destruction accounts for 31 %. Session handling sums up to 92 % of the total chaincode invocation time on QEMU offering significant potential for performance improvement. On the Raspberry Pi 3B session establishment accounts for 70 % of the chaincode invocation time and session destruction accounts for 23 %, summing up to 93 % for session handling during the total chaincode invocation time. Without the overhead from session handling during each chaincode invocation, FOC would benefit from significantly improved throughput and lower latency that scale with the number of clients.

4.8.2 Small-scale Deployment

In this evaluation FOC is deployed over 25 devices and two platforms in order to measure the energy efficiency and scalability for up to 128 clients of the system. During 5 minutes clients continuously issue transactions with either read or write requests. FOC is set up so that clients are evenly distributed among peers by connecting each client server with one peer. Each component in the FOC network is executing on its own server except for clients. The baseline is set up identical to section 4.8.1 in which all components are executed on peers in the normal world without using gRPC.

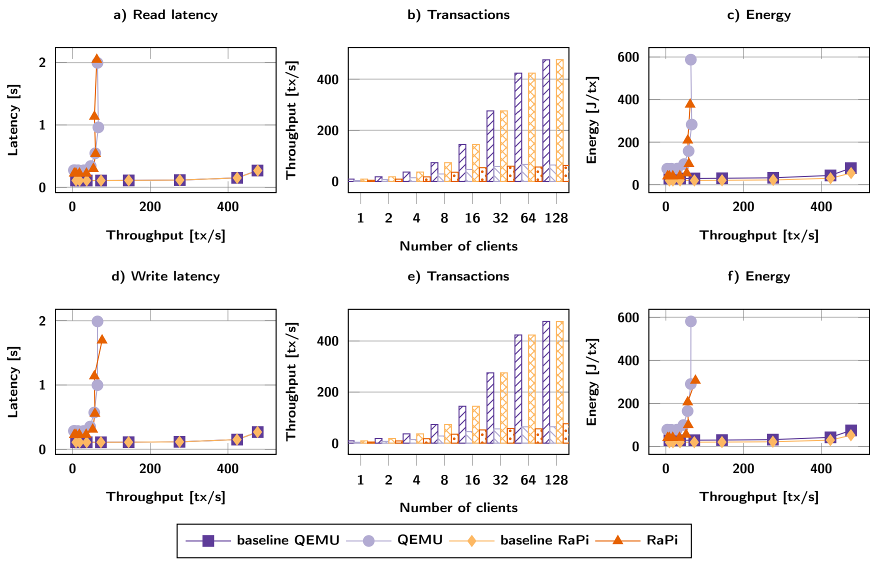


Figure 4.3: Throughput-latency, number of transactions and transaction energy plots for read/write invocations. Top row are read transactions, bottom row are write transactions.

Figure 4.3 shows the scalability and energy efficiency of FOC with the top row examining read requests and the bottom row examining write requests. The baselines on the hardware and with QEMU are very similar which indicates that the chaincode design is simple and not introducing any significant overhead. Measurements are capturing the effect of shielding chaincode execution inside the secure world since gRPC is also not interfering with the measurement as found out in the previous section 4.8.1. It should be noticed that baseline measurements are barely beginning to saturate the system with 64 and 128 clients. When shielding the chaincode using TrustZone, the system saturates between 8 to 16 clients at around 65 tx/s. This result is in line with the $1.2\times$ scalability of throughput observed in section 4.8.1 indicating that at most one client per server is invoking a chaincode on the TrustZone-enabled nodes. The oversight in the FOC implementation of not reusing the TEE session serializes all chaincode invocations and introduces a significant overhead due to session management operations. The impact of shielding chaincode execution by TrustZone is completely overshadowed by this overhead. Under ideal conditions reusing sessions can scale the throughput by a factor of $4\times$ resulting in a theoretical system throughput of 260 tx/s for FOC with 32 concurrent chaincode invocations. This would correspond to about half the throughput of the baseline which is a significant overhead for shielding chaincode execution, but it also includes communication overhead with the peer over gRPC. In the unsaturated region when shielded by emulation with QEMU latency increases by a factor of $2.5\times$ over baseline for read requests and by a factor of $2.6\times$ for write requests. Latency for read requests in the unsaturated region increase by a factor of $2.0\times$ and for write requests by a factor of $2.1\times$ on the Raspberry Pi. Throughput and latency for read requests increases by a factor of $1.04\times$ over write requests with

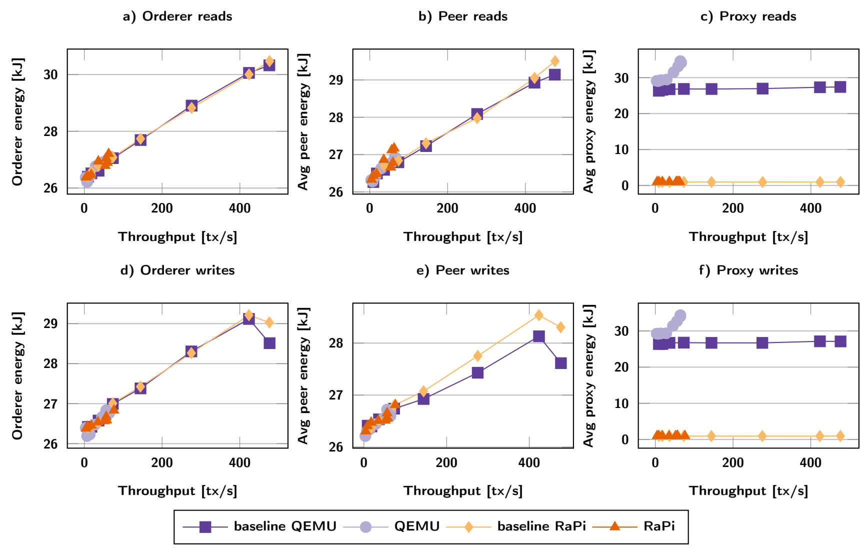


Figure 4.4: Energy consumption of nodes in the Fabric network. Top row are read transactions, bottom row are write transactions.

QEMU. On the Raspberry Pi throughput and latency of read requests increase by a factor of 1.02 by comparison with write requests.

Figure 4.4 presents the energy consumption of the three different types of nodes in FOC with the top row representing read requests and the bottom row representing write requests. During measurements two auxiliary servers record power consumption of 15 nodes in the FOC network that is mapped to benchmark results for obtaining the energy consumption. Energy consumption is stable across nodes and increases with an increasing number of clients. For baseline measurements energy increases with the number of clients (Figure 4.4-[a,b,d,e]) until approaching saturation point of the system on orderer and peer nodes. As a result, with 128 clients throughput is no longer increasing at the same rate as for lower numbers of clients. For write requests energy consumption is falling off due to increasing latency and OSs lowering OPPs of servers. Baselines on proxy nodes are roughly constant (Figure 4.4-[c,f]) because these devices are idling while chaincodes execution happens on peer nodes. Due to the limited scalability of FOC energy consumption of orderer and peer nodes does not increase beyond 8 clients (Figure 4.4-[a,b,d,e]) except for proxy and chaincode execution (Figure 4.4-[c,f]) emulated on QEMU. The overhead from session management as observed for latency and throughput is also increasing energy consumption when executing with QEMU due to many I/O bound requests and limited resources for storage and networking. Energy consumption is also increasing on the Raspberry Pi (Figure 4.4-[c,f]) but barely visible on the low power platform. This observation is better visible in Figure 4.3-[c,f] where energy cost per transaction is increasing proportionally to the energy consumption of the QEMU instance. Energy consumption of proxy nodes beyond saturation point rises at steady rates of 1.25 × with QEMU and by 1.05 × on Raspberry Pi. There is no significant difference between read and write requests on the two platforms except for 128 clients on Raspberry Pi were the energy is lower because of a lower latency.

4.9 Discussion

Confidential smart contracts are challenging for developers to implement as TAs given the limitations imposed by TrustZone and OP-TEE. FOC is a prototype for blockchain networks designed for IoT and based on Fabric for executing confidential smart contracts as chaincodes inside the secure world of TrustZone. The chaincode execution has been separated from the Fabric architecture in order to isolate it in the secure world, shield it from compromised as well as malicious entities and minimize the increase of the TCB. Remaining components are considered untrusted, with the exception of the ledger, and execute in the REE. The FOC prototype leverages the modularity of Fabric for distributing components over several heterogeneous devices. In that regard FOC is a continuation of the idea of a distributed OS as stated by Fabric. The possibility to execute confidential smart contracts on embedded, mobile and IoT devices enables many new innovative solutions for example in the finance, medical and supply chain sectors or the industry 4.0.

The distinct architecture of Fabric decouples the ordering phase from the execution phase which facilitates scalable blockchain networks. Combined with the modularity of components makes it possible to deploy the blockchain network over heterogeneous platforms adapted to the requirements of applications. For example, ledgers that need to maintain the world state of blockchains require large amounts of space and are better placed in clouds or data centers. Peer nodes or even the concept of proxy nodes can be deployed at the edge, where they are closer to the users in order to reduce latency. Executing smart contracts directly on IoT devices brings DLT at the frontier of the Internet and allows to physically interact with blockchains. Sensor data from IoT devices can be linked to conditions in smart contracts. Such applications are in particular interesting in the supply chain, where IoT devices could control environmental conditions during the transport of goods. For example, some sensitive chemicals or pharmaceuticals can decompose when exposed for longer periods of time to higher temperatures. Many challenges lie ahead by pursuing this vision and mature solutions have yet to be developed and explored.

Although the FOC prototype is not negatively influenced by the world switching overhead, because of the relatively high communication latency between chaincodes and the ledger, the initially stated issue of network-enabled secure IoT services remains open. A network protocol designed to minimize the network traffic and storage cost would be very beneficial for secure IoT services by mitigating the negative impact of the world switching overhead. However, IoT services in general can benefit from the previously described characteristics. By reducing the network traffic, less energy needs to be spend for data transmission which reduces the energy consumption and extends the battery lifetime. The following chapter is describing the attempt to design and implement such a protocol.

5

Developing Energy-Efficient and Scalable IoT Protocols

5.1 Introduction

IoT is generating vast quantities of data that need to be processed and stored. Processing and storing the data locally is not possible because of limited resources available on IoT devices. Data is either offloaded to the edge or the cloud which requires the data to be transferred over the Internet. The amount of network traffic is continuously increasing with every newly deployed IoT device. It is estimated that in 2022 80 ZB of data will be generated [264] by digital devices and that 6 % of the 4.8 ZB global Internet traffic is due to IoT data [78, 79]. More than half of the 28.5 billion total Internet connections will be IoT devices [78, 79] that on average send 323 B/s.[1] Recently deployed 5G networks are going to triple the amount of IoT traffic compared to 4G networks [78]. Over time connectivity will be less of an issue for IoT devices and now the focus should shift towards data management. The challenge will be on the innovation of novel solutions that make future IoT traffic manageable and reduce the total energy consumption that is due to offloading.

Compression algorithms appear as an obvious solution to large data volume problems and cloud storage offerings. However, the limited resources available on IoT devices make it either difficult or even impossible to implement common compression algorithms such as LZW [306] or DEFLATE [96]. Efficient compression algorithms tend to be memory demanding, whereas lightweight compression algorithms are more memory-efficient but cannot achieve the same high compression ratios. IoT services disseminate packets of small sizes that have to be compressed individually due to memory limitations. Compression ratios of common compression algorithms are degrading as data sizes are decreasing. The evaluation of a lightweight compression algorithm for wireless sensor networks [197] demonstrates this behavior. Other approaches consider using lightweight compression algorithms which trade off data accuracy for compression ratio [72, 100, 103, 135, 256]. Introducing errors in the data might be acceptable in some use cases, but it is not acceptable for smart metering or industrial sensing.

Network deduplication [283] approaches the problem differently by substituting recurring byte sequences with shorter hash values. These hash values are then used on the receiving side for looking up the intended byte sequence of the substituted data. This means that source and sink nodes need to agree

[1] These estimations are based on numbers before the COVID-19 pandemic. The Internet traffic has since then changed a lot and the mentioned numbers might no longer be accurate.

on a mapping of hash values to byte sequences. Asymmetric caching [271] proposes to use a cache at the source node in order to deduplicate byte sequences and which is updated by feedback from the cache of the sink node. The feedback contains cache updates to be applied by the source node for increasing the matching probability of byte sequences. Manifest feedback [315] is a file communication system that has been successfully deployed for deduplicating network traffic and saving network bandwidth. The source node splits the data into chunks and computes for each chunk its hash value. These hash values are then compared to the hash values in the cache of the source node and deduplicated in case of a match. Unmatched hash values are sent to the sink node where further matching analysis is conducted. The sink node responds with feedback on query information and manifests of matched chunks. In a similar way it is possible to apply deduplication to video streams [148] by merging independent streams and labeling them with identifiers. Deduplication benefits in this situation from large data volumes with deterministic content that can make use of local caching with large hash sizes on routers.

IoT services and in particular secure IoT services are being deployed on platforms with constrained resources. On these platforms IoT data is generally highly volatile and usually directly transmitted to edge devices before being forwarded to clouds for storage. Computational resources are primarily being used by IoT services for aggregating and transmitting data. Network protocols have to account for limited resources and provide solutions that reduce the network traffic while making efficient use of the available computational resources. Especially spacial and temporal correlations can be exploited with IoT data in order to improve compression ratios and reduce network traffic. Combining all these necessary properties in a single protocol requires novel techniques that are specifically designed around IoT use cases. These techniques need to be both energy-efficient and scalable for taking into account battery-powered IoT devices that form networks with large numbers of nodes. IoT devices and services alike benefit from higher availability through extended battery lifetime and improved energy efficiency by using a customized protocol.

Hermes is proposed as a new protocol for IoT services that combines several of the previously mentioned approaches with GD [298]. GD is a novel lightweight compression technique which can be implemented on resource limited IoT devices in contrast to compression algorithms. Data is split into chunks and GD is applied in order to obtain bases and deviations on source nodes. For each basis a corresponding fingerprint is computed. Each fingerprint is paired with its deviation and sent opportunistically to intermediate nodes. Intermediate nodes are located at the edge and it is expected that they have some local caching capabilities. Incoming fingerprints are matched against a local cache that uses asymmetric caching with sink nodes located in the cloud. For any unmatched fingerprints the intermediate nodes will request the corresponding bases and forward them to the sink nodes. This mechanism is similar to manifest feedback and reduces network traffic by minimizing the number of transmitted bases. The sink nodes assemble bases from all IoT streams into knowledge bases that are synchronized with caches on intermediate nodes.

This chapter describes the design, architecture, implementation and evaluation of Hermes. Section 5.2 summarizes the state of the art and presents protocols commonly used by cloud-based IoT services. Related work on deduplication and GD is outlined in section 5.3. The general concept of GD is explained in

Table 5.1: Protocols used for cloud provider IoT solutions

Protocol	AWS	Azure	GCP	IBM	Oracle	Tencent
AMQP	✘	✔	✘	✘	✔	✘
CoAP	✘	✘	✘	✘	✔	✔
HTTP	✘	✘	✔	✔	✔	✔
HTTPS	✔	✔	✘	✘	✘	✔
MQTT	✔	✔	✔	✔	✔	✔
WebSocket	✘	✘	✘	✘	✔	✘
XMPP	✘	✘	✘	✘	✔	✘

section 5.4 with references to its formal principle. The architecture of Hermes is specified in section 5.5. Hermes is evaluated locally on a single node and in a small scale deployment with 16 Raspberry Pis in section 5.6. The chapter concludes with a discussion in section 5.7 on the evaluation and its impact on the state of the art.

5.2 State of the Art

Cloud providers offer fully integrated IoT solutions for seamless interaction with other cloud services. Table 5.1 provides a set of commonly used protocols for IoT services and if cloud provides support them. It should be noticed that HTTP, HTTPS and WebSocket rely on data exchanged in specific data formats such as CSV or JSON. The XMPP protocol is available on one cloud provider, but usually it is used for instant messaging and therefore not further described in this section. The remaining protocols are being described in this section in order to provide a comparison to Hermes and clearly highlight the differences in their design. These protocols were not necessarily designed to be used by IoT services but are standardized and industrially well established protocols. Many protocols are relying on Message-Oriented Middleware (MOM) and establish peer-to-peer networks that stand in contrast to hierarchical network topologies that are required by cloud IoT services. Each protocol is designed for different use cases and satisfies some subset of properties essential to IoT services. The lack of a protocol that is at the same time energy-efficient and scalable is demonstrating the need for Hermes.

5.2.1 Advanced Message Queuing Protocol

The Advanced Message Queuing Protocol (AMQP) [238, 299] is an open standard asynchronous messaging protocol. Only proprietary implementations of asynchronous messaging systems with incompatible interfaces existed at the time AMQP was conceived, which lead to the need of a protocol with an open standard. AMQP is a MOM relying on exchange services that provide message queues for asynchronous delivery. Nodes establish connections to each other in order to form a peer-to-peer network for setting up sessions and exchange messages over links between source nodes and target nodes. Messages are produced by clients and processed by other clients while traversing the network via several brokers, which store messages in queues. AMQP encodes messages in a binary protocol over TCP with support for various data types and data structures. The protocol can either be secured using TLS or SASL [209] for which AMQP provides framing support.

5.2.2 MQ Telemetry Transport

MQ Telemetry Transport (MQTT) [239] is a standardized protocol in which clients are connected to a broker that routes application messages between clients. The protocol establishes a pub/sub system in which clients can publish application messages under topic names, subscribe to topics of interest and unsubscribe from subscribed topics. A server operates the application message broker to which clients publish application messages or subscribe to receive new application messages on a specific topic. The broker filters topics against client subscriptions in order to disseminate application messages in the network. The protocol supports integer data types such as two bytes, four bytes and variable bytes. Strings are UTF-8 encoded within text fields and binary data is limited to 64 KiB for MQTT packets. MQTT requires reliable transport of packets based on TCP connections on top of which protocols such as TLS or WebSocket can be used. Various authentication methods have built-in support for the protocol. The threat model of the protocol considers compromised devices, access to data at rest on clients and servers, protocol timing attacks, DoS attacks, man-in-the-middle attacks and packet injection. Counter measures against these threats include authentication of users, authorization for accessing server resources, integrity and privacy guarantees of application messages. If hardware support on IoT devices is available, MQTT proposes to use AES [92] and otherwise to use ChaCha20 [44] implemented in the software.

5.2.3 Constrained Application Protocol

The Constrained Application Protocol (CoAP) [280] is specifically designed for IoT devices and resource-limited networks. Interactions between application endpoints are based on alternating request and response messages similar to a client-server model. CoAP can easily interface with HTTP and provides support for Uniform Resource Identifiers (URIs) and media types. The Representational State Transfer (REST) architecture has been adapted to the constrained resources of IoT devices and provides a subset of the functionalities as Constrained RESTful Environments (CoRE). Discovery of services and resources are additional features built into the protocol. At the transport layer CoAP relies on the unreliable and connectionless UDP in order to minimize the complexity of protocol stack implementations. The protocol provides a confirmation mechanism with tokens to make message exchanges over UDP reliable. Unconfirmed messages are retransmitted using an exponential back off algorithm. It is not excluded that messages are duplicated because confirmations did not arrive or were delayed. In this case the protocol requires that each duplicated message has to be confirmed. Security guarantees are added by using the DTLS protocol which is essentially the equivalent of TLS but for datagram-based protocols such as UDP.

5.3 Related Work

5.3.1 Data Compression in Sensor Networks

Data compression and dual prediction schemes exploit spacial and temporal correlations in network traffic in order to conserve network bandwidth and reduce energy consumption. Both schemes are simultaneously applied on an IoT network [158], so that traffic between IoT devices and intermediate nodes is compressed by dual prediction schemes whereas traffic between intermediate nodes and sink nodes uses data compression schemes. Dual prediction schemes are based on machine learning models that are trained on neural [146] and long short-term memory [143] networks. The data compression schemes include principal component analysis [310], non-negative matrix factorization [110], truncated-singular value decomposition and discrete wavelet transform. It should be noticed that these data compression schemes are lossy and affect the accuracy in order to improve compression ratios. In the case of data compression the schemes can be further classified as either distributed or local depending on the origin of data. Distributed schemes compress data originating from multiple nodes in the network and exploit spacial and temporal correlations. Local schemes only take temporal correlation into account from data that originates on the same node. The compression schemes collect data and from blocks of specific sizes before compression. Compression data blocks are then sent from intermediate nodes to sink nodes, where they are decompressed to recover a lossy version of the original data. Evaluation was carried out on simulation of the Intel Berkeley Research Lab data set [150] by using temperature values only. The machine learning models accurately predict temperature values on IoT devices and reduce network traffic by up to 46 %. The different lossy compression schemes demonstrate that IoT data has high compression potential. Larger block sizes reduce transmission sizes but increase latency and may reduce accuracy of data. It is claimed that energy can be saved by combining dual prediction with data compression schemes without supporting measurement results.

5.3.2 Deduplication

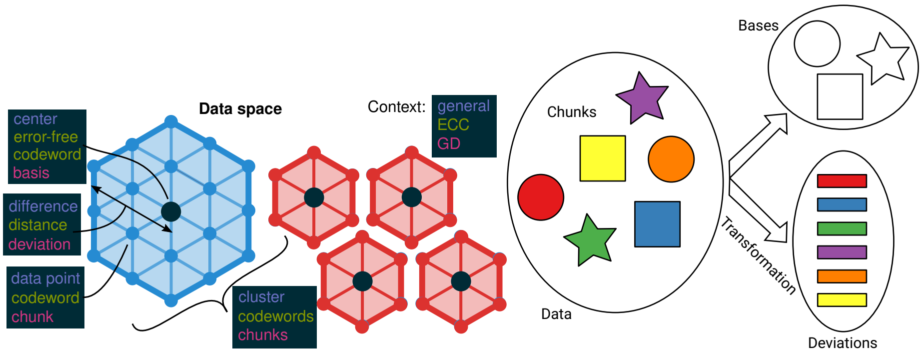
Deduplication is a technique that is used in networks [283] for reducing traffic and also in file systems [214, 251] for reducing storage requirements of files on disks. This technique can either be source-based or target-based depending on whether the technique gets applied on the client side or on the server side [134]. In an evaluation study [214] with 857 different file systems, distinct chunks with identical content were identified and a single copy was stored. On average these file systems contained 225 000 files with an average file size of 318 kB which were structured in 36 000 directories. The evaluation compares whole file, fixed block sizes and Rabin fingerprinting [260] as deduplication schemes. Rabin fingerprints identify identical content in files that might have been shifted due to deletions or insertions. For every chunk in the system, independent of the utilized scheme, a fingerprint is computed and these fingerprints are then compared with each other. The comparison of fingerprints can be sped up by using caches or Bloom filters [46], but the evaluation did not make use of any possible optimizations. Rabin fingerprinting achieved the best compression ratio with 40 % followed by fixed block-level deduplication with 30 %. File-level deduplication achieves the lowest space savings of 22 % but is claimed to have lower impact on computational complexity than Rabin fingerprinting and fixed block-level deduplication. Consequently, it is concluded that whole-file deduplication in combination with sparseness is a highly efficient and optimal scheme for lowering storage consumption.

5.3.3 MinervaFS

MinervaFS [233] is a FUSE-based file system [4] that leverages generalized deduplication for saving space on non-volatile storage devices. A POSIX-like interface is exposed by the FUSE layer for various file system functions. Underneath this layer is the generalized deduplication layer that provides several transformation functions and handles deduplication. Transformation functions implement various ECCs that separate data chunks into basis and deviation pairs. Deduplication is then applied on the bases by using fingerprinting. The lowest layer is the kernel file system on top of which FUSE file systems are mounted. Any POSIX-compatible file system can be used as backend in order to read files from and write files to disk. The file system is compared against ext4 [201], SDFS [7] and ZFS [314] with various combinations of compression and deduplication features enabled. On average MinervaFS reduced the storage space by 27 % and up to 64 % compared to ZFS on real datasets. In particular for read-heavy workloads MinervaFS has a 16 % performance lead over ZFS. Compared to compression techniques such as Gzip, MinervaFS only requires 20 % more and 2.75 × less storage than ZFS. The transformation function used by MinervaFS for the evaluation was Hamming code. The compression potential of generalized deduplication is not exhausted and other transformation function could further optimize storage costs.

5.3.4 ZipLine

ZipLine [297] leverages generalized deduplication in network switches to compress and decompress data at line speed of 100 Gbit/s for getting higher network throughput. The network switches are based on the Tofino platform [84] and their data plane can be programmed using the P4 programming language [50]. Built-in support for Cyclic Redundancy Checks (CRCs) is exploited on the Tofino platform to implement transformation functions based on Hamming codes. The evaluation compares various setups of ZipLine such as pre-populated lookup tables or dynamic learning. Results indicate that both approaches are viable options and can achieve compression and decompression rates of up to 7 Mpkt/s at line speed with little to no loss in throughput or compression ratio. For dynamic learning latency analysis reveals that it takes (1.77 ± 0.08) ms to register new bases in lookup tables before generalized deduplication can be applied to packets. The transmission size of synthetic and real datasets could be reduced by up to 89 %, respectively 90 %, which compared to gzip is about 20 % higher. Furthermore, it is not possible to implement a DEFLATE algorithm as used by gzip with P4 on the Tofino platform.



(a) Concept of GD (b) Principle of GD

Figure 5.1: The concept and principle of GD visualized

5.4 Generalized Deduplication

Generalized Deduplication (GD) [298] is a novel lossless data compression approach leveraging ECCs. Identical and similar data chunks are deduplicated through transformation functions, that systematically cluster the data space as in Figure 5.1a. Data chunks represent data points in the data space that can be associated to specific clusters without the need to compare the current chunks to previously registered chunks. There are several functions that transformation functions can make use of in order to compute the center of clusters and the distances of data points to those centers. The output of transformation functions is referred to in GD as basis-deviation pair, where the basis corresponds to the center of the cluster and the deviation to the distance, respectively the error. From an abstract point of view the basis corresponds to a shape and the deviation to a color as represented in Figure 5.1b. Classical deduplication methods are special cases of GD where the deviation is always zero, because each data point in the data space is its own cluster.

5.4.1 Fingerprinting

A simple way to deduplicate two data chunks is to compare each byte of the data chunk for equality. This can easily be done for smaller data chunks but with increasing sizes the process becomes computationally more expensive. Instead, deduplication methods rely on fingerprints that are obtained by applying hash functions, such as SHA-1 and SHA-256, or checksums, such as CRC32 and MD5. Fingerprints have identical sizes and can be much shorter than data chunks. The problem with fingerprints is that several data chunks can map to the same fingerprint and cause a collision. This means that if two fingerprints are identical, deduplication methods still need to compare the data chunks in order to ensure that they are identical and eliminate the possibility of false positives. Fingerprints can be stored conveniently in hash tables which allows for fast lookup and enables the use of caches in distributed deployments.

With GD fingerprints are generated for and assigned to bases. Each basis can then be stored exactly once, whereas each deviation is paired with the fingerprint of the basis. Instead of storing directly the basis-deviation pair output of transformation functions, fingerprints can provide further storage savings. These storage gains are also exploitable in networking, where network bandwidth can be saved by sending fingerprint-deviation pairs instead of basis-deviation pairs.

5.4.2 Pre-processing

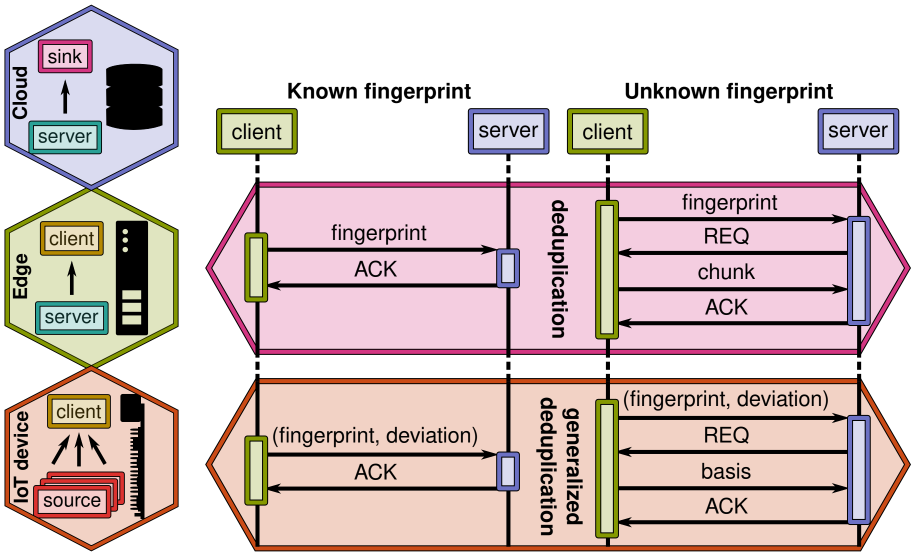
Before applying transformation functions to data chunks there a couple of pre-processing methods worth considering to further improving compression ratios. Specifically in IoT use cases data chunks might consist of multiple time series or sensor data records. *Delta encoding* keeps the first record in a data chunk unchanged while replacing all following records by the difference to the first record. This method is particularly efficient when records have constant or small variation. For example, records with two-byte number representation could be reduced to single-byte difference if the variation is small enough. The problem with delta encoding is that differences can either be positive or negative, which reduces the number range by one bit that is needed in order to indicate the sign. Another mechanism that avoids the sign problem is *offset removal*. In a first step the minimum record in data chunks is identified. Then, all records are substituted by their offset to the minimum record. Offset removal has the advantage that offset ranges are twice as large as difference ranges but the disadvantage is that the minimum record needs to be stored separately. The pre-processing methods were considered specifically for multi-record data chunks, but it is certainly possible to apply these methods across multiple data chunks.

5.4.3 Transformation Functions

There are several functions that can be used in order to map [234] data chunks to bases and deviations. This work limits the scope of transformation functions to ECCs and more specifically to Hamming codes. Another type of ECC that could be used are Reed-Solomon codes [263]. In the context of ECCs, data chunks become codewords, bases become messages and deviations hold bit position where in the codeword bits deviate from the error-free codeword. In order to obtain the bases and deviations, ECC decoding functions have to be applied on data chunks. Data chunks are obtained by applying ECC encoding functions on the bases and changing bits according to deviations.

Hamming Codes

Hamming codes [133] are a type of ECCs that can be used as transformation function. Let m be the number of parity bits, then codewords have a bit length of $n = 2^m - 1$ and messages have a bit length of $k = 2^m - m - 1$. However, in order to make codewords more manageable with byte-addressable computers, Hamming codes with two parity bits are not considered and codewords have power-of-two length. The additional bit can either be considered as a regular data bit that is not covered by the parity bits or it is simply ignored. In any case, the additional bit is considered to be part of the parity bits such that actually codewords have length $n + 1$ and message have length $k + 1$. Hamming codes can correct single-bit errors and detect double-bit errors. With respect to GD this means that any data chunk has at most a distance of one to the



(a) Architecture (b) Packet exchange

Figure 5.2: Hermes architecture with network hierarchy and packet exchange between client and server threads.

error-free data chunk, or the radii of clusters is always one. The position of the bit deviating from the error-free codeword is encoded in a syndrome vector of length m. Without getting too formal, the syndrome is identical to the column of the parity-check matrix, in which an error has occurred. If the syndrome is zero, then no error has occurred in the codeword and the codeword is error-free.

5.5 Architecture

The Hermes protocol is implemented based on hop-by-hop schemes. In hop-by-hop schemes intermediate nodes have to store state information for each flow. Hop-by-hop protocols are typically found in sensor networks where data is aggregated in tree-like network topologies. Ideally, Hermes is implemented as an application protocol on top of a hop-by-hop transport protocol. Secure Data Aggregation Protocol [311] would be such a protocol, but it has unfortunately been implemented on the application layer. With its best knowledge and belief non of the proposed hop-by-hop protocols have been fully implemented nor standardized. For this reason Hermes relies on manual network topology setup using traditional end-to-end transport protocols such as TCP and UDP. Hermes distinguishes between three types of nodes in a network, which are source nodes, intermediate nodes and sink nodes as shown in Figure 5.2a.

Three modes of operation allow nodes to either send raw data, perform classical deduplication or GD. Modes enable flexible networks in which resource limits of nodes are taken into account without compromising the benefits of GD. Nodes sending raw data act as simple pass-through without looking up fingerprints in caches and doing deduplication or GD which is useful for resource constrained devices. Deduplication mode limits nodes to perform classical deduplication but has the advantage these nodes can be combined with nodes sending with raw data. For example, deduplication can be delegated to intermediate nodes while source nodes send raw data. The same kind of interaction is possible with nodes operating in GD mode. This means that nodes operating in pass-through mode can be connected to nodes either in deduplication or GD mode. With Hermes GD stages can be separated and distributed over multiple nodes in the network.

Source nodes, as the name implies, are devices where data originates. When source nodes join a network, they send out a message to an intermediate or sink node containing a unique stream identifier together with a structure describing the data layout of the stream. The receiving node locally registers the stream identifier together with the structure. From this point on source nodes can start streaming data packets into the network. Multiple source threads aggregate data from sensor or through other means and forward the structured data to client threads. The client threads pull structured data from message queues and wrap them in Hermes packets and send them over TCP to their destination. The packet exchange between client and server threads is show in Figure 5.2b.

Intermediate nodes store the structure of the data streams and keep a cache of known basis fingerprints. They are located between source nodes and sink nodes and indirectly connect these two types of nodes. Upon receiving a new data message, a server thread extracts the fingerprint and looks it up in the cache. Then, the server thread either requests the basis for the fingerprint from the sending node before forwarding it over client threads to the next node, or acknowledges the data message and awaits newly incoming data messages. This approach ensures that bases are forwarded exactly once from source nodes to sink nodes. This hop-by-hop scheme leverages GD in order to minimize data streams to sink nodes hosted in the cloud. Depending on the number of streams and the cache size, a significant part of network bandwidth throughput can be shifted to source nodes as close as possible and free up network bandwidth closer to sink nodes at the cloud.

Sink nodes are located at the root of the tree-like network topology. They operate server threads similar to intermediate nodes that respond to incoming data messages. The backend assembles a knowledge base of all bases disseminated in the network and constructs lists of fingerprint-deviation pairs for each data stream. Server threads in sink nodes do not require caches and can instead directly interface with the knowledge base for querying bases.

Figure 5.3 is describing the schematic implementation of GD inside Hermes nodes. Initially some data, for example from a sensor, is provided as codeword to the lightweight compression algorithm. The codeword is in a first step decoded into the message and its parity bits by applying the transformation function, *i.e.*, Hamming code. Then the syndrome of the codeword is computed by multiplying the codeword with the parity-check matrix H of the transformation function and masking the parity bits of the product. This step can be optimized for IoT devices by multiplying only the parity sub-matrix with the codeword instead of the entire parity-check matrix H. The syndrome provides the index into the syndrome table, from where the position of the error within the message can be retrieved. The basis is obtained by flipping the erroneous bit in the message. Depending on the knowledge base if the basis fingerprint is contained or not, either the basis concatenated with the syndrome or the fingerprint of the basis concatenated with the syndrome are provided as output.

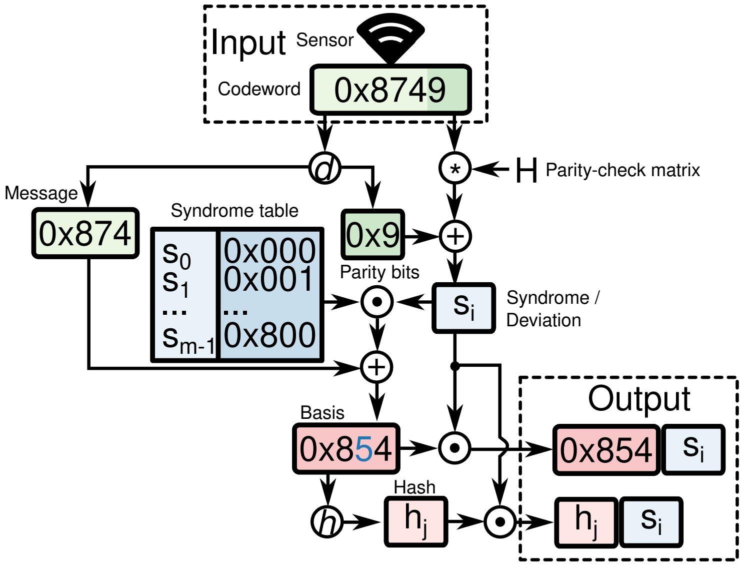


Figure 5.3: Implementation of GD in Hermes

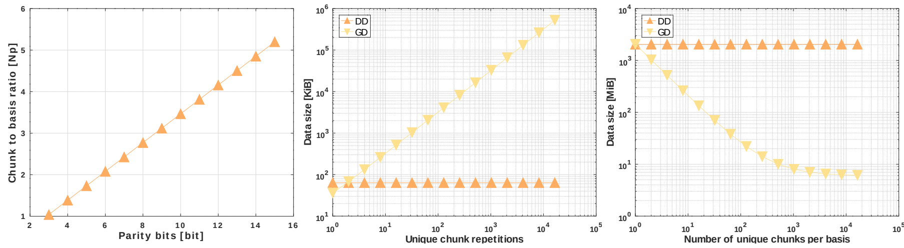
5.6 Evaluation

5.6.1 Setup

The prototype implementation of Hermes is evaluated on a cluster of 16 *Chaumont* devices equipped with a Raspberry Pi PoE-HAT [262] and powered by 802.3af Power-over-Ethernet [210] from an Ubiquity Networks UniFi USW-48P-750 switch. Each Raspberry Pi has the beta bootloader 2019-12-03 installed for PXE boot deployment of Raspbian Buaster Lite 2020-02-13 with Linux v4.19.97 on a 32 GiB SanDisk Extreme microSDXC UHS-I card. The Raspberry Pis are used as source nodes that over Gigabit Ethernet connect to a sink node on a *Pilatus* machine. Intermediate nodes are not included in the setup to avoid any interference from processing overhead. Power consumption of the sink nodes is recorded by a *Kaiseregg* auxiliary machine. The clocks of all machines are synchronized using NTP in order to relate the power consumption to the statistics collected during benchmark runs.

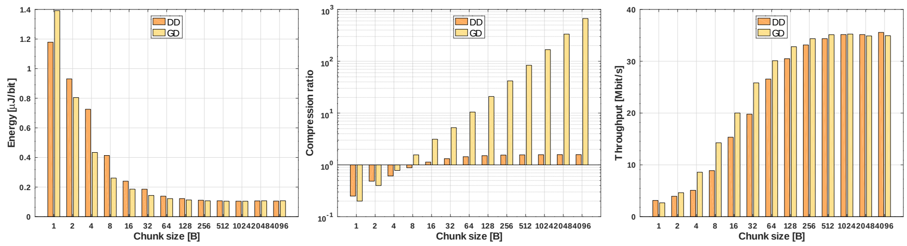
5.6.2 Classical and Generalized Deduplication

The impact of data set properties are evaluated in a first comparison between classical deduplication and GD, in particular how repetitive data chunks and the mapping of data chunks to bases affects compression ratios for both approaches. These properties are best validated using synthetic data sets where the properties can be well controlled. In order to generate synthetic data sets, a generator creates a number of random messages of length k. These messages are then encoded to append the parity bits and yield error-free codewords. With respect to Hamming codes, the generator introduces a single bit flip for every error-free codeword to obtain candidate codewords. Each candidate codeword is then decoded in order to verify that it still maps to the same basis. If the



(a) Chunk to basis ratio (b) Repetitive chunks (c) Unique chunks per basis

Figure 5.4: Comparison of classical and generalized deduplication

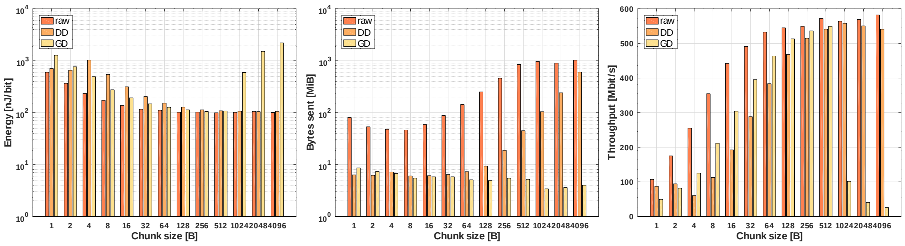


(a) Energy per bit (b) Compression ratio (c) Throughput

Figure 5.5: Micro-benchmark on Raspberry Pi 4B

verification failed, then the codeword is reverted, a new bit flip is introduced and verified until the codeword maps to the same basis. The generator can be parameterized by the number of unique bases and the number of unique data chunks.

Figure 5.4a describes in general the ratio given in nepes of data chunks to basis for consecutive numbers of parity bits. The number of data chunks mapping to a basis increases with the number of parity bits, respectively the data chunk size. As the number of parity bits increases, the compression ratio also increases. Deduplication is the favored approach over GD for data sets with many repetitive data chunks as seen in Figure 5.4b. The problem of GD is that despite deduplicating repetitive data chunks to the same basis, deviations have to be stored as well. In the case of a data set with 2 B data chunks the storage overhead is so significant that storage requirements keep increasing for GD. If instead data sets are composed of unique data chunks mapping to the same basis as in Figure 5.4c, then it is still possible with GD to deduplicate data chunks, whereas classical deduplication can no longer deduplicate any data chunks. This last data set consists of a little more than a million data chunks where the number of unique data chunks mapping to the same basis is increased. The data size is approaching its minimum for as long as the number of unique data chunks approaches the data chunk size. Afterwards the data size stagnates as the storage requirements for deviations become more important.



(a) Energy per bit (b) Network traffic generated (c) Throughput

Figure 5.6: Macro-benchmark on Raspberry Pi 4B cluster.

5.6.3 Micro-benchmark

The micro-benchmark evaluates the local impact of classical deduplication and GD on the Raspberry Pi for various generated synthetic data sets. Figure 5.5 shows the relation of the chunk size to the energy, compression ratio and throughput. In Figure 5.5a GD is more energy-efficient than classical deduplication for small chunk sizes except single bytes. As the chunk size increases both approaches level out requiring almost identical amounts of energy, despite the computational overhead with GD. Beyond chunk sizes of 8 B GD achieves better compression ratios than classical deduplication as outlined in Figure 5.5b. At the largest chunk sizes between 1024 to 4096 GD has a two orders of magnitude higher compression ratio than classical deduplication. This observation is a direct consequence of the number of chunks mapping to a basis, shown in Figure 5.4a. The throughput for either approach on a single thread peaks at 34 Mbit/s as in Figure 5.5c. For chunk sizes smaller than 32 B there is a significant overhead limiting the throughput. This evaluation includes reading data from memory, applying transformation functions, computing and looking up fingerprints. It appears likely that frequent memory accesses are reason for the overhead.

5.6.4 Macro-benchmark

A simple Hermes network with 16 Raspberry Pis is deployed for the macro-benchmark evaluation of the protocol. The baseline for the evaluation are source and sink nodes in pass-through mode where they send and receive raw data. For every chunk size a dataset of exactly 1 GiB in size was generated. In order to compare the results in Figure 5.6 of the macro-benchmark to the micro-benchmark in section 5.6.3 one source and one client thread are executed per Raspberry Pi. There is a computational overhead for 64 B chunk and smaller sizes with a clear difference to the baseline. It should be noticed that for all three modes energy is amortized until reaching chunk sizes of 64 B. As mentioned in the micro-benchmark section 5.6.3, this amortization is likely related to minor computational overhead from masking smaller than word-size chunks and memory accesses, since cache lines on the Cortex-A72 are 64 B wide [30]. Another reason for the amortization could be the time resolution of power consumption statistics by the UniFi PoE switch. Power consumption is updated every 4 to 5 seconds which reduces the accuracy in particular for shorter benchmark runs.

The energy for 1024 B to 4096 B chunk sizes, which is larger than average IoT data size, suddenly rises in GD mode. In this range, the Raspberry Pi reach their memory limit due to additional data copies that are needed while preparing Hermes packets. Figure 5.6b shows the network traffic that is generated by the different modes. Under ideal conditions, the baseline should always be close to 1 GiB. The dip observed for small chunk sizes in the range of 1 B to 64 B is because of lost packets as the network gets hammered with too many small packets. The network traffic stabilizes in the range of 64 B to 512 B as fewer packets are sent across the network and network congestion is reduced. With GD generated network traffic is reduced from 9 MiB to 5 MiB as chunk sizes increase from 1 B to 512 B, while for classical deduplication network traffic starts to increase significantly beyond chunk sizes of 128 B. Between 64 B to 128 B seems to be the sweet spot where the number of unique chunks has the upper hand on the number of duplicate chunks. The shape of the throughput in Figure 5.6c is very similar to the shape in Figure 5.5c with the exception of the drop in throughput for GD when reaching the memory limit. Dividing throughput numbers by 16 closely matches the throughput numbers measured for a single Raspberry Pi.

5.7 Discussion

Hermes is a new protocol that leverages GD as a novel lossless compression technique which is particularly suited for IoT services. GD can exploit spacial and temporal correlations of IoT data in order to achieve compression ratios higher than classical deduplication and which can even exceed compression algorithms [125]. This is made possible by generalizing deduplication and introducing transformation functions which systematically cluster similar data points and which avoid comparison to other data points. These properties are very beneficial for IoT devices which have limited resources and where commonly used compression algorithms cannot be implemented.

The GD approach is decomposed by Hermes into individual modules that can be distributed over the network. For example, resource limited source nodes generate IoT data that is directly passed through an intermediate node which applies the transformation function and deduplicates the data. This highly flexible design does not compromise compression ratios or network bandwidth and enables energy-efficient IoT deployments where IoT devices can spend more time in idle power states as there is fewer data to be send across the network. The scalability of Hermes networks strictly depends on the storage capabilities of intermediate nodes at the edge. More network bandwidth becomes available close to clouds, at the core of the Hermes network, as more data can be cached at the edge of the network. Scalability is the principal driving force for high compression ratios across the entire network with GD.

Lossless compression is a key property of GD compared to other compression techniques for IoT deployments. However, it is important to understand the limits of GD which have been addressed in the previous evaluation. GD reaches its maximum potential when data has small variation and is accumulated at close distance. Under these circumstances data is mostly composed of unique values that map to the same base value. If data is composed of mostly repeated data points, then it is better to switch to classical deduplication approaches in order to avoid storage overhead from deviations. Current IoT cloud solution use protocols besides CoAP that have not been designed for resource limited IoT devices. Compression and deduplication mechanisms cannot easily be integrated into these protocols because specific packet exchanges are required or other limitations have to be considered. Packet header sizes are another important factor for reducing network bandwidth and saving energy. Hermes attempts to take all these properties for IoT services into account in order to provide an energy-efficient and scalable protocol. GD is a powerful technique if used correctly which addresses many of current and future challenges in IoT. There is a lot more potential in this novel technology that is waiting to be explored.

Undervolting is the last technique that remains to be explored to specifically improve the energy efficiency of IoT services in general. Thus far previous chapters have demonstrated how energy efficiency in combination with scalability and security is improved for secure as well as network-enabled IoT services. TEEs provide hardware-based solutions for secure IoT services that surpass software-based alternatives, such as homomorphic encryption, in terms of performance and energy consumption. Network-enabled IoT services can take advantage of spacial and temporal correlations exploited by the Hermes protocol in order to reduce network traffic and allowing for longer idle periods as IoT devices needs to spend less time transmitting data. Both approaches are complementary and are, depending on the requirements, not necessarily applicable to IoT services in general. The following chapter is exploring in particular undervolting as an technique to improve the energy efficiency of cloud instances where, among many other services, IoT services are deployed.

6

Undermining Reliability With Energy Efficiency

6.1 Introduction

The power consumption of processors depends on the manufacturing processes and transistor count. Processor architectures are broadly divided into RISC and Complex Instruction Set Computer (CISC) architectures. RISC architectures follow a simple design philosophy with a reduced ISA that generally results in programs with more instructions. CISC architectures include translation processes which decompose instructions into sequences of microcodes. By assembling microcode to sequences, more instructions can be expressed which enriches the ISA of CISC architectures. As a result, RISC architectures can be implemented using fewer transistors than CISC architectures, which reduces the overall power consumption of processors.

Energy is an important resource of clouds. Cloud providers are continuously upgrading their infrastructures to make use of energy-efficient devices and to adapt to customer and market needs as well. The cloud market is dominated by CISC architectures, in particular the x86 architecture, but recent cloud provider offerings begin to include the ARM architecture [1, 8, 23, 40, 63] that is based on a RISC architecture. The ARM architecture was originally designed for low-power embedded and mobile devices before a new microarchitecture was introduced by ARM in order to expand the architecture to the cloud market.

Public clouds are used in particular by large-scale IoT services for processing and storing aggregated IoT data. The limited resources available on IoT devices and on the edge are restricting the amount of data that can be processed and stored on these devices. Instead, IoT services are designed to offload the data to clouds where resources can be scaled depending on the requirements. The processes deployed on clouds are critical to IoT services and provide as endpoints valuable insights into the nature of the collected data. If the obtained results would get corrupted, then incorrect conclusions could be drawn from the data. Data corruption can occur through soft faults when processors operate at critical supply voltages. Under these conditions instructions executed by processors are no longer reliable and can lead to erroneous results. Critical supply voltages are reached by undervolting processors such that the critical path delay of instructions is roughly equivalent to the inverse processor frequency. Soft faults are hard to detect and can be mitigated through specific hardware or independent and redundant software executions.

This chapter explores the risks incurring when operating ARM processors at reduced voltage margins. More specifically a cloud infrastructure equipped with low-power ARM processors undervolted by an excessively economic and malicious cloud provider is outlined in an exemplary manner. Raspberry Pis are used in order to study the behavior of undervolted instances based on the ARM architecture. Malicious cloud providers and users exploiting undervolted instances to leak sensitive information [71, 288] are not considered, but instead this chapter is answering the following two research questions:

1. Can cloud instances be undervolted in a stealthy way by cloud providers?

2. Can undervolted instances be detected by a cloud user?

The risk of operating processors at reduced voltage margins are summarized in section 6.2 A study motivating undervolting and the use of ARM processors in the cloud is given in section 6.3. Section 6.4 presents the state of the art while section 6.5 introduces related work. The threat model is described in section 6.6, where on the one hand the cloud provider is trying not to reveal the undervolted state of the cloud infrastructure by setting up a strong adversary model called the *Scrooge Attack*, whereas on the other hand the cloud user is trying to detect any undervolted instances. The detection method of the cloud user outlined in section 6.7 is then evaluated in section 6.8 on bare-metal and container instances. Section 6.9 provides answers to the previously asked research questions.

6.2 The Risks of Reduced Voltage Margins

Before the introduction of the ARM Neoverse microarchitecture an ARM microarchitecture for server-grade platforms did not exist. OEMs and semiconductor manufacturers needed to either customize application-grade ARM microarchitectures or develop custom server-grade ARM microarchitectures. The AWS Graviton processor [1] is an example of a customized application-grade ARM microarchitecture used in AWS data centers. Custom-made ARM SoCs were offered in form of business card instances by the cloud provider Scaleway [195]. There are even collocation offers by cloud providers to host, for example, Raspberry Pis [295] in order to free resources on server-grade hardware. Because ARM processors are low-power processors by design, they also dissipate less heat and therefore require less cooling energy. Cloud providers can pack more ARM devices in a rack than x86 devices due to these thermal inconveniences. Consequently, energy and costs spent for cooling cloud infrastructures can be reduced by using RISC architectures.

Modern processors have DVFS systems installed, which allow developers to individually regulate the frequencies and voltages of processors. These mechanisms were highly efficient as long as the majority of power consumed by processors was dynamic. With the end of Dennard scaling [94] processors began consuming more static power than dynamic power and the returns of these mechanisms were diminishing [179]. Hence, new approaches were needed to improve the energy efficiency of processors. Adjusting voltage margins is one approach among many for reducing energy consumption of processors.

Voltage margins are determined by semiconductor manufacturers based on the process variation and are applied uniformly for the entire processor batch. The savings from voltage margins are less than those of DVFS systems and appear negligible on single devices, but are important at scale and in particular

Table 6.1: List of current and past server-grade ARM processors offerings by cloud providers.

Processor	ISA	Cloud provider
Ampere Altra	ARMv8.2+	Equinix, Oracle
Ampere eMAG 8180	ARMv8	Equinix
AWS Graviton	ARMv8	AWS
AWS Graviton 2	ARMv8.2	AWS
Marvell custom made SoC	ARMv7	Scaleway
Marvell ThunderX	ARMv8	Equinix, OVH Labs, Runabove, Scaleway
Marvell ThunderX2	ARMv8.1	Microsoft Azure

to cloud providers where they accumulate for each deployed processor. Reducing the voltage margin of processors requires undervolting. Undervolting primarily conserves energy and influences operating temperatures and core aging.

Cloud providers may have several reasons to undervolt their cloud infrastructure, either for energy saving or for cost saving reasons. Independent of the intents of the cloud providers, there is a limit to the degree of undervolting that processors can support. Undervolting cannot be applied arbitrarily and comes at the cost of processor reliability. When a supplied voltage is insufficient to drive the frequency of the processor, timing violations occur resulting in soft or hard errors. Soft errors are recoverable and result in process failures, while hard errors end in crashes and unavailable cloud instances. Unavailable cloud instances caused by incorrect setups are typically covered by SLAs, which cloud providers are avoiding as much as possible. A scrooge may certainly be willing to take such risks by favoring cloud instance setups with soft errors over hard errors to overcome SLA protections.

Undervolting has been exploited in recent attacks [166, 228] for gaining access to sensitive information. However, the cloud user is confronted by a different threat model, in which roles have been reversed. Cloud instances can be purposely undervolted by a powerful attacker, *i.e.*, the cloud provider, such that cloud users have to depend on potentially unreliable cloud instances. The options at disposal of cloud users are restricted in order to uncover such hard to detect errors, without physical access to instances or being able to directly change supply voltages or frequencies. If access to frequency and voltage is restricted, then cloud users can only indirectly influence them through CPU load and core temperature. Kernel schedulers adjust frequencies and supply voltages pairwise as OPPs of CPUs, based on the amount of remaining work, which is given as CPU load. By increasing CPU loads on a processor, core temperatures of the processor will rise as more heat is dissipated. With increasing core temperatures the resistivity of the circuitry of the processor increases as well, which is compensated by rising supply voltages using hardware or firmware. Cloud users can manipulate frequencies and supply voltages through CPU load and core temperatures in order to force soft errors by erroneous computations.

6.3 ARM Processors in the Cloud

ARM processors have in recent years found their way into clouds and data centers. Table 6.1 lists server-grade ARM processors with the ISA and cloud

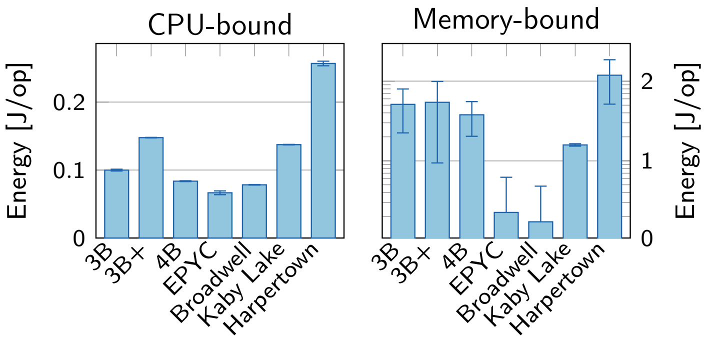


Figure 6.1: Energy comparison of Raspberry Pi model B and x86 servers for CPU-bound and memory-bound workloads.

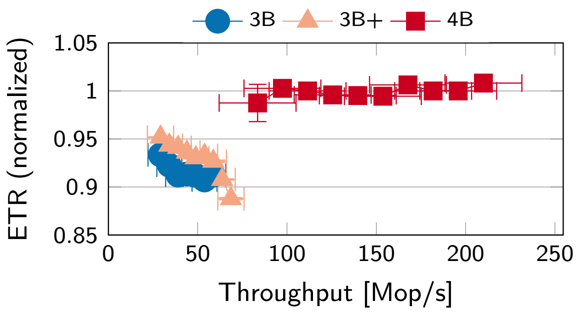


Figure 6.2: Normalized ETR with absolute errors for undervolted Raspberry Pi model B platforms.

providers. The continuous performance improvements of low-power ARM processors has made this architecture a compelling alternative to the dominating x86 architecture. Latest server-grade ARM processors have $2\times$ to $4\times$ more cores than current x86 processors. Figure 6.1 compares the energy consumption per operation of Raspberry Pi models 3B, 3B+ and 4B to x86 servers for CPU-bound and memory-bound workloads. The same CPU-bound and memory-bound source code is compiled to binaries for both architectures. An AMD EPYC and Intel Xeon Broadwell, Kaby Lake and Harpertown processors were used as x86 processors. The energy consumption per operation between ARM and x86 are comparable even with the large difference in power consumption. There are no major differences between CPU-bound operations, but memory-bound operations of all Raspberry Pi models consume more energy. The difference is due to small cache sizes and low memory transfer rates. For both workloads, all Raspberry Pi models are more energy-efficient than the older Harpertown microarchitecture.

Undervolting is applicable to low-power ARM processors with significant energy savings. Figure 6.2 shows the normalized ETR ratio at minimum supply voltage for the Raspberry Pi model 3B, 3B+ and 4B. The 3B and 3B+ are undervolted by $-75\,\mathrm{mV}$ while the 4B is undervolted by $-15\,\mathrm{mV}$. Undervolting directly influences the energy consumption of operations without negatively affecting throughput. Energy efficiency improved due to undervolting on the 3B and 3B+ between 5 % to 13 % and on the 4B between 0 % to 3 %. The downside of undervolting is the risk of introducing soft and hard errors due to timing violation. Such errors can be avoided by carefully analyzing the guardbands of processors. Secure services deployed in clouds need to be particularly aware of undervolting, because confidentiality and integrity guarantees can be broken [166, 228].

6.4 State of the Art

6.4.1 Guardband Identification

Benchmarks are commonly used for identification and evaluation voltage margins. The SPEC CPU2006 has been used in [247] for identifying the voltage margins on low-power and high-performance Intel processors. With the help of the machine check architecture [151] in Intel processors recoverable and unrecoverable hardware errors can be detected. The core voltages of the processors were adjusted between benchmark runs and during evaluation errors were monitored for determining crash points. Based on this method, it was observed that voltage margins are 10 % on low-power and between 12 % to 15 % on high-performance processors from nominal voltage. By reducing voltage margins up to 20 % of power can be saved on both processors and operating temperatures can be reduced by up to 25 %. It is important to notice that the cores in processors do not have identical behavior and that there are variations due to process variation or positioning of the core within the processor layout.

In a similar way the guardband of a server-grade ARM processor has been evaluated [246]. Without access to an architecture capturing hardware errors, the system was characterized by a software-based approach that relied on performance counters and linear regression for feature selection. Severity was introduced as a metric in order to characterize the behavior of cores when undervolted based on a classification of features. The severity metric is then expressed as weighted sum of the selected features. During the evaluation of the severity metric using SPEC CPU2006 and bwaves benchmarks, three distinct voltage regions were identified. In the safe region no errors occur whereas in the unsafe region abnormal behaviors are experienced, finally before in the crash region the system fails. An interesting observation is that silent data corruptions are occurring at higher voltage levels rather than corrected errors. Even on low-power architectures it is possible to save up to 19 % energy by severity characterization of cores without having identified their minimal voltage level. Reducing performance by 25 % enables even higher energy savings of up to 39 %. By extending this approach using statistical methods for different feature selection methods [162], it is possible to very accurately predict voltage margins with similar energy gains. The correlation between the hardware counters and the minimal voltage level combined with the severity is the key to accurate prediction models.

6.4.2 Micro Viruses

Micro viruses [245] provide a different approach for determining voltage margins on processors. These viruses stress specific components of processors, such as caches and pipelines, in order to test the limits of DVFS and accurately determine the minimum voltage level [37, 104, 316]. The time needed for identifying voltage margins using this approach is reduced by more than one order of magnitude with little to no loss in accuracy compared to benchmark approaches. Benchmark approaches run several workloads for covering as many components of processors as possible which is a very time consuming process. Power viruses, such as SYMPO [111], MAMPO [112], MPrime [128] or stress-ng [6], are benchmarks which maximize power consumption of processors. However, power viruses cannot be used for voltage margin determination because power consumption and minimal voltage levels are not correlated [245]. Minimal voltage levels are depending on the critical path of instructions executed by processes. It is possible that two distinct processes have the same power consumption but different minimal voltage levels due to the nature of their instructions.

6.4.3 Online Monitoring Daemon

The problem with benchmark or micro virus for guardband identification is that minimal voltage approaches are error prone and can lead to system failures [244] because they do not capture all possible workloads in deployed systems. Oftentimes minimal voltages are determined individually per core and do not take into account multithreaded workload deployments. As the thread count is increasing, dependency of minimal voltages on the nature of the workload is weakening and frequency as well as core allocation have stronger influences. Instead, a daemon monitoring workloads online could make dynamic adjustments to core allocation for workloads, voltage and frequency in order to minimize energy consumption. This approach is minimally interfering with performance of workloads due to the daemon monitoring concurrently all executing processes. Evaluations on two server-grade ARM processors using 35 different workloads have demonstrated that this approach can save up to 25 % energy at the cost of up to 3 % in performance. In comparison to static approaches like guardband minimization, this approach saved $2\times$ more energy. A major contribution in energy savings is in particular due to dynamic frequency adjustments and core allocation of workloads.

6.4.4 ParaMedic and ParaDox

ParaMedic [16, 17] exploits heterogeneous cores in processors in order to provide error resilience. Main cores have an out-of-order pipeline and are superscalar, while checker cores have an in-order pipeline, operate at lower frequency and are area as well as power efficient. Multiple checker cores are assigned to a single main core in the ParaMedic architecture. Computations on main cores are at runtime divided into segments of contiguous instructions. At the end of every segment a checkpoint is taken, which launches a checker core with the latest checkpoint for recomputing the segment and validating the state of the main core. When the state of the checker core diverges from the state of the main core, an error has been detected. The state is then rolled back to the last checkpoint and execution is resumed. In order to allow recomputing states of the main cores, a log of all load and store instructions with the memory address and value is kept in SRAM. Main cores have to wait at checkpoints for checker cores to become available. All execution is duplicated and the architecture has the advantage to run without ECC in the main core.

ParaDox [18] is an extension to ParaMedic with control over frequency and voltage to influence energy consumption and error rate of the architecture. The architecture can dynamically adjust to the error rate by increasing the length of segments at low error rates or decreasing the length of segments at high error rates. In case the voltage is set too low for a workload and error rates increase, the system can dynamically adjust voltages to bring the system back into the safe region of operation and reduce error rates. ParaDox keeps track of the highest voltage setting at which errors occurred for every application. In a first phase ParaDox targets an error-seeking region for applications in order to define voltage reference points when an error occurs. These reference points define a voltage region at runtime for applications with optimal power consumption but where errors can still happen. Every 100 errors these reference points are re-set for readjusting voltages in case applications have changed their operational behavior. Voltage spikes in the system can lead to further errors and are avoidable by temporarily reducing the frequency when errors occur. However, in some situations reducing the frequency can result in timing violations and voltage adjustments are unavoidable. ParaDox has been evaluated in a simulation and estimations indicate a 22 % reduction in power consumption at error rates of two orders of magnitude higher than for ParaMedic.

6.5 Related Work

6.5.1 Undervolted Wireless Sensor Networks

Nodes in wireless sensor networks can benefit from undervolting with longer uptime and availability of the network. For an undervolted wireless sensor network hardware prototype using an ATmega1284p [173] the impact of temperature on microcontrollers and transceivers is evaluated. The problem with some microcontrollers is the dependency between frequency and voltage. Microcontroller frequencies generated from linear electronic oscillators depend on the voltage level. Frequencies are lower than nominal frequencies by undervolting which can affect certain instructions and render time critical processes or asynchronous communication unreliable. No failures appear until undervolting reaches 10 % of nominal voltage when failure rate sharply rise to up to 100 % failures. The microcontroller clock rate can be recalibrated to a nominal rate in order to mitigate errors and make the system reliable. It was observed that lower temperatures require higher supply voltage while higher temperatures require lower supply voltages. However, raising temperatures for reducing supply voltages is not ideal because of accelerated core aging [265]. The transceiver saved about 7 % energy by undervolting and no change in received signal strength indicator or link quality indicator at 1 m distance was noticed. Combining all energy savings from undervolting microcontroller and transceiver under various temperatures, up to 42 % of energy per clock cycle could be saved.

6.5.2 Plundervolt

Plundervolt [228] is a software-based attack against Intel SGX enclaves in which faults are injected by undervolting CPU cores. The attack exploits an undocumented Intel Core voltage scaling interface in order to target instructions executed by SGX enclaves and make them unreliable. Several x86 instructions were evaluated for fault injections such as arithmetic and bit-wise instructions of which multiplication instructions faulted. It is assumed that the length of the critical path and the complexity of the circuitry are possible reasons that faults can be injected. Similar observations were made for other instructions that presumably have complex circuitry like for example AES-NI. A program containing an infinite multiplication loop could be terminated by invalidating the loop test condition through undervolting the CPU core and injecting a bit flip error. This bit flip error was occurring outside and inside of SGX enclaves. Further investigation of the bit flip error revealed that either between one to five bits were flipped or all most significant bits were flipped. By changing the program and limiting the number of multiplications, higher undervolting needs to be applied in order to inject fault during program execution. Process variation between two identical processors resulted in different conditions under which faults could be injected. Temperature is affecting undervolting so that with lower temperatures higher undervolting needs to be applied for fault injection. Using these insights the authors could extract cryptographic keys from SGX enclaves or redirect EPC page lookups to attacker controlled memory pages by faulting array index addresses. In order to mitigate this attack, Intel has disabled the voltage scaling interface by a Basic Input/Output System (BIOS) update. This attack not only broke the confidentiality guarantees of SGX it was also the first attack to break the integrity guarantees.

6.5.3 Dynamic Undervolting at Runtime

Minimum voltages can be individually determined for applications of which the highest among the minimal voltages is statically applied to voltage domains. This approach, however, is not optimal because the same undervolting is applied to all applications independent of their workload. More energy can be saved by dynamically adjusting undervolting for applications based on the workload at runtime. Context-aware dynamic undervolting mechanisms at the OS level can take workloads into account and provide application-specific undervolting at runtime. CADU [199] is a prototype that can also differentiate between user- and kernel-level code for individual undervolting adjustments. The prototype is implemented as Linux kernel extension, performing only voltage scaling and no frequency scaling on unmodified applications. Applications are characterized offline in order to determine their safe minimum voltage, using binary search between nominal voltage and $-300\,\mathrm{mV}$ undervolting. The minimum voltage setting is considered safe when the application did 10 consecutive executions without failures. Context switches and system calls trigger callback functions for CADU to adjust undervolting to the next thread to be executed. In case the undervolting has to be reduced, *i.e.*, the supply voltage needs to be raised, CADU waits for the voltage domain to reach the appropriate supply voltage before scheduling the thread. Power savings are split into 24 % due to static undervolting, 4 % due to inter-application variability and 2 % due to intra-application variability. The evaluation indicates that power consumption of the processor can be reduced by 6 % to 30 % compared to static undervolting approaches. An overhead between 0.5 % to 0.8 % is introduced by the prototype on average across several benchmarks due to decision taking and waiting for voltage domains to reach target voltages.

Alternative software-based approaches integrate undervolting functionalities in the DVFS governor of OSs [172]. Pipelines are frequently interrupted by events in particular when processes interact with the hardware. As a result, safe minimum voltages for processes change dynamically at runtime. The approach determines offline first minimum supply voltages individually for all cores and then combined core minimum supply voltages. In a second phase processes are profiled using performance counters in order to determine their interactions with the CPU. Results acquired from these two phases are fed into a data model for single-threaded execution and a data model for multi-threaded execution for determining safe minimum supply voltages. The two models are then deployed to the governors which dynamically adjust supply voltages based on the model predictions and processor load. Problems with single-threaded workloads for determining minimum supply voltages are optimistic margins. Therefore, it is necessary to profile voltage margins also for multi-threaded workloads in order to capture complex interactions. Compared to the standard P-state DVFS governor, the model-based approach saved 42 % more energy on the Skylake microarchitecture and 34 % on the Haswell microarchitecture.

6.6 Threat Model

6.6.1 The Scrooge Cloud Provider

The cloud provider is considered to have full access to the infrastructure and the physical machines. Physical machines are undervolted by the cloud provider and use ARM processors for additional power savings. The undervolting is done statically so that the minimum voltage is configured in the safe region and as close as possible to the critical region for instances to run stable without failures or errors. Firmware configurations are not accessible by cloud users. The cloud provider is running a PITM attack on firmware requests. Any read requests to the firmware are intercepted and manipulated by the cloud provider in order to hide the undervolted configuration of the infrastructure and which is not made oblivious to users.

6.6.2 The Curious Cloud User

Curious cloud users intend to uncover any suspicious, hidden or obfuscated activities going on in the background of instances. Access to cloud instances is exclusively provided over remote connections to cloud users. Under regular setups cloud users can simply lookup instance configurations or query the firmware for identifying undervolted states. However, when configurations or firmware queries are forged, cloud users have a much more difficult stand to uncover and proof that instances are undervolted. User space lacks a tamper-proof message exchange with the firmware that is confidentiality and integrity protected and cannot be changed when traversing the kernel layer. This mechanism is essential

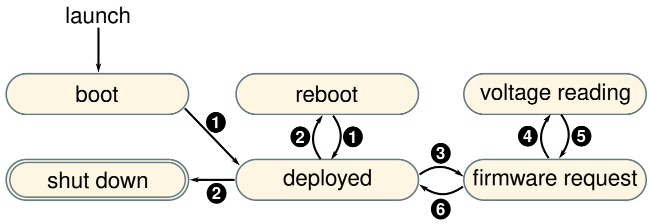


Figure 6.3: State machine describing cloud provider actions to obfuscate undervolted states of cloud instances

for detecting undervolted processors.

Kernel warnings or kernel panics during system boot or at runtime are hints to cloud users that instances are undervolted and operating in the critical region. These warnings and panics are not specific and can contain all kinds of errors resulting from undervolted processors. In particular when instances require more time than necessary in order to become available, then there is a possibility of a failed boot attempt due to the kernel crashing. The boot process is typically logged by the OS and can be accessed by system administrators. It is assumed that the cloud provider can tamper with boot logs and cloud users cannot rely on the information that is stored in these logs. The TCB is limited to the software supplied by the cloud user.

6.6.3 The Scrooge Attack

The cloud provider makes undervolted ARM instances available to its cloud users that are indistinguishable from instances operating at nominal voltages. Configurations, tools and firmware hide the undervolted state of instances and presents cloud users forged values mimicking nominal voltage settings. This requires mechanisms intercepting requests targeting the voltage setting in which undervolted values are substituted by nominal values.

Figure 6.3 describes which actions the cloud provider needs to take during the cloud instance lifecycle in order to prevent the undervolted state from leaking to the cloud users. The cloud provider has to make sure that newly launched cloud instances by cloud users are deployed with the undervolted configuration. Once the undervolted configuration has been applied by the firmware, the configuration should no longer be accessible when the cloud users log into their instance. During the boot process the undervolted configuration has to be substituted by a nominal configuration (Fig.6.3-❶) before reaching the deployed state and granting access to cloud users. Hidden or obfuscated system services can do this substitution during the boot process. An alternative approach using trusted OSs or auxiliary devices [2] can do the substitution during early boot without involving the OS. The file system cloud users find on their system might not be the file system they launched. Configurations, firmware and tools could have been substituted during the boot process depending on the undervolting mechanisms. The nominal configuration has to be substituted for the undervolted configuration (Fig.6.3-❷) when instances are shut down or rebooted.

For deployed instances any read requests for core voltages have to be intercepted at runtime and substituted by plausible nominal voltage values. A kernel driver located between user space and hardware will be communicating with firmware or accessing model-specific registers. Requests can then be intercepted either by tools in user space or by kernel drivers (Fig.6.3-❸). Intercepted requests are forwarded (Fig.6.3-❹) and undervolted voltage values are returned (Fig.6.3-❺). The tool or kernel driver then substitutes the returned undervolted voltage value by some forged nominal value, for example, by adding the voltage offset to the returned value. After substitution, the answer containing the forged nominal value to the read request is returned back to user space (Fig.6.3-❻). Read requests could be delegated to TAs under the control of the cloud provider, which is a more costly alternative for the cloud provider but provides higher confidentiality.

The outlined interception mechanisms need to be adjusted if cloud users are allowed to deploy their own kernels. Under these circumstances the cloud provider can no longer intercept read requests in user or kernel space. Instead, interception mechanisms have to be integrated into the hypervisor and substitutions need to be performed in a similar way as described for kernel drivers.

The cloud provider is assumed to be very powerful and to make use of the previously described mechanisms in order to obfuscate the undervolted state of cloud instances. From the cloud providers perspective the efforts are practicable and highly beneficial. Cloud users are confronted with an untrustworthy situation in which they can never be sure to receive genuine voltage values from read requests. Without an integrity protected message exchange with firmware or physical access to cloud instances, cloud users are unable to verify the authenticity of voltage values and have to assume that all values are forged.

6.7 The Detection Method

The threat model described in the previous section 6.6 does not allow cloud users to define specific conditions for the detection method to uncover the scrooge attack. Cloud users have to rely on parameters the cloud provider has little influence on. The processor load and the package temperature are closely connected parameters to which the cloud user has limited margins for adjustments. Simple CPU-bound programs can be used to put the processor load under maximum load. These programs should have similar properties to power or micro viruses and specifically target one component of the processor. The multiplication program used in [228] is ideal for this purpose because not only does it maximize the load and power consumption but also can monitor the program itself. Self-monitoring can be implemented for arithmetic computations by verifying the result. By repeating the same arithmetic computation in a loop, load is maximized and faults can immediately be identified from incorrect results. The processor is operating at the highest frequency under maximum load. Under these circumstances lots of heat is dissipated by the processor and package temperatures are rising. Faults are injected with rising temperatures as a result of timing violations. The highest probability to inject faults is reached when temperatures are no longer rising. Firmware throttling is enabled at high temperatures to prevent the processor from taking damage.

For the CPU-bound program two random numbers are generated at the beginning and then multiplied in an infinite loop while altering the position of the multiplier and the multiplicand. After each multiplication operation the results are verified because it was observed in [228] that the position of the multiplier and the multiplicand can lead to faulting instructions. Instructions fault depending on the complexity of the circuitry that implements the instruction. Complex instructions tend to fault more easily than simple instruction. The detection method is customizable under this aspect to either generate faults or provoke faults in concurrently running processes. The later behavior is favorable because it allows the detection method to run until multiple failures in system relevant processes render the instance unavailable. Undervolted cloud instances are identifiable by continuously provoking faults in system processes until reaching the crash point. Cloud users can execute one CPU-bound process per processor core in order to maximize load and heat dissipation while monitoring the instance for failures.

The success rate of the detection method depends on the remaining voltage margin after undervolting and the cooling system used in the data center. Wide voltage margins require higher temperatures for faults to occur and *vice versa*. Cooling systems are less problematic because temperature rates are primarily influenced and consequently the runtime of the detection method is prolonged. The better the cooling system the simpler should the instruction be in the CPU-bound program. As the runtime of the detection method increases, so does the probability to inject a fault in one of the threads of the detection methods.

The cloud provider is limited in preventing the detection method from revealing the scrooge attack. On the one hand the cloud provider could configure a soft limit temperature that would throttle instances and limit fault injections. This counter measure could backfire in case of demanding regular workloads in which case cloud users would change to competing cloud providers with better performing instances. On the other hand the cloud provider could increase the voltage margin but this would defy the purpose of the scrooge attack and reduce profits from savings on the electricity bill. Despite having a very strong adversary in this threat model, there are still ways to enable detection methods. What is in particular interesting with this scenario is the simple design of the detection method compared to all counter measures put in place by the cloud provider to mount the scrooge attack.

Instance failures are covered by the SLA as long as the cloud provider is at fault because of some misconfiguration or because assured QoS parameters are not met. SLAs might distinguish between single-instance deployments and multi-instance deployments. In case of single instance failures cloud users need to provide evidence that sufficiently support the claims in order to be covered by the SLA. For multi-instance deployments all instances have to become simultaneously unavailable for being eligible to make use of the SLA. System log files provide the necessary supporting claims demonstrating that the faulting process was not the primary process running on the instance due to non-selective fault injection. In some cases the faulting processes might even be under control of the cloud provider. All this becomes a lot trickier with multi-instance deployments, in which case many factors make simultaneous instance failures non-deterministic. For example process variation plays into the hands of cloud providers and helps to obfuscating the undervolted setup. Given the complexity of simultaneous failures for multi-instance deployments, the scope of the detection method is limited to single-instance failures.

6.8 Evaluation

The Raspberry Pi is used as platform for mimicking ARM-based cloud instances. First, the experimental setup describes how to automate a temperature-based guardband analysis and defines the configurations to run the evaluations under realistic data center conditions. The throttling behavior of the firmware needs to be understood before any attack parameters can be determined. For each model safe, critical and failure voltage regions are characterized by a temperature-based guardband analysis. Probabilities for successfully identifying undervolted instances are determined under various configurations by studying the faults that occurred during the guardband analysis. Afterwards, appropriate parameters are determined by the cloud provider in a series of benchmarks in order to mount the scrooge attack and profits from undervolting are estimated. An analysis of the benchmark results can also be used by cloud users for choosing suitable types of programs for the detection method. The section concludes with an evaluation of the detection method deployed on undervolted bare-metal and container instances.

6.8.1 Experimental Setup

For the evaluation the latest three Raspberry Pi models 3B, 3B+ and 4B are used. A single SD card is used for all three models with Raspbian Buster as OS and firmware versions from June 2020. All measurements are taken in an air conditioned room at $(24 \pm 1)\,^{\circ}\mathrm{C}$ under realistic data center conditions. The Raspberry Pis are connected over Ethernet to the network and have a SSH daemon running for remote connections as well as the NTP enabled for clock synchronization. An auxiliary machine monitors the Raspberry Pi over UART besides which no other peripherals are connected in order to minimize any interference. The auxiliary machine records the power consumption during the evaluation over Bluetooth from the PowerSpy. NTP is enabled on the auxiliary machine for correlating the power consumption to the benchmarks running on the Raspberry Pi. The previously described setup has been fully automated under manageable effort with the help of a network-capable power strip.

6.8.2 Firmware Throttling

The Raspberry Pi 3B+ is the only model for which the firmware is defining a soft limit temperature of $60\,^{\circ}\mathrm{C}$. By default the 3B+ operates at higher supply voltage and is overclocked despite using the same SoC as the 3B but with better voltage regulators and a better thermal design. Soft throttling could be a viable mitigation technique against the detection method depending on the performance loss. What is known from the official documentation is that frequency and voltage of the SoC are reduced in order to decrease heat dissipation. For the evaluation a multiplication benchmark is used and the CPUFreq governor is set to *performance*. The exact behavior is not described in the documentation, so table 6.2 presents empirically determined parameters. A consistent behavior is observed for all overvoltage levels, where CPU core frequencies f_{arm} are reduced by 200 MHz and supply voltages V_{arm} are lowered by 106 mV which is the equivalent of four levels. There are two exceptions to the voltage stepping of 25 mV which are from nominal level $-1 \rightarrow -2$ with -32.2 mV and from soft

Table 6.2: Soft limit (SL) firmware throttling on the 3B+

OV level	V_{arm} [V]	f_{arm} [MHz]	V_{arm}^{SL} [V]	f_{arm}^{SL} [MHz]
0	1.3750	1400	1.2688	1200
-1	1.3500	1400	1.2375	1200
-2	1.3188	1400	1.2125	1200
-3	1.2938	1400	1.1875	1200

Table 6.3: Limit temperature (L) throttling on the 3B and 4B

Model	V_{arm}^{L} [V]	f_{arm}^{L} [MHz]	f_{core}^{L} [MHz]
3B	1.2813	$\{1034, 1087, 1141, 1195, 1200\}$	$\{400\}$
4B	0.8500	$\{1000, 1500\}$	$\{333, 500\}$

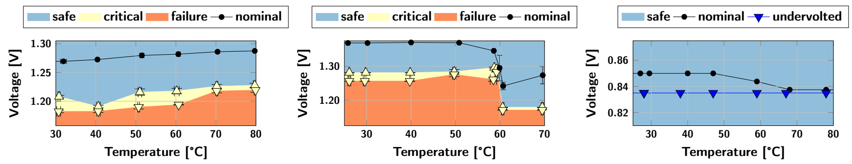
limit level $0 \rightarrow -1$ with $-31.3\,\mathrm{mV}$. The soft limit essentially downgrades the 3B+ to a 3B.

Thermal runaway of the SoC is prevented by the firmware at the limit temperature of 85 °C for all models. The 3B+ is not included in this evaluation, because the soft limit throttling makes it extremely difficult to reach the limit temperature [294]. Again, a multiplication benchmark is used and the CPUFreq governor is set to *performance* for determining the parameters shown in table 6.3. An interesting difference between the soft and the limit temperature is that the limit temperature only reduces the frequency while the supply voltage stays the same for both models. This affects only the dynamic power consumption of the SoC leaving the static power consumption untouched. The frequency columns f_{arm}^{L} and f_{core}^{L} indicate sets of frequencies that the firmware will apply in order to reduce the temperature. CPU core frequencies f_{arm}^{L} of the 3B are reduced in steps of 54 MHz. The 4B reduces the CPU core frequency in a single step by 500 MHz and also reduces the GPU frequency f_{core}^{L} by 167 MHz.

Both the soft and limit temperature throttling are reducing the CPU core frequency by 15 % on the 3B and 3B+. The limit throttling on the 4B reduces the CPU core frequency even by 33 %. These are significant performance losses that will limit the fault injection rate of the detection method. However, the throttling does not appear to be a viable mitigation technique as it severely impacts performance and could affect highly demanding cloud workloads.

6.8.3 Temperature-based Guardband Analysis

The temperature-based guardband analysis is used to dynamically determine minimum voltages for the safe, critical and failure voltage regions. This approach is capable of capturing voltage adjustments made at runtime by an AVS system. The analysis is divided into the following stages. First, the OS is booted with an undervolted SoC. In a second stage, the temperature of the SoC is adjusted either actively or passively in order to reach the starting temperature of the benchmark. As soon as the starting temperature is reached the CPUFreq governor set to *performance* and a benchmark is executed for determining the minimum voltages and monitoring the failure rate of the undervolted configuration. This procedure ensures that the benchmark is starting at a well defined temperature and at constant maximum frequency. Finally, when the benchmark



(a) Raspberry Pi 3B (b) Raspberry Pi 3B+ (c) Raspberry Pi 4B

Figure 6.4: Temperature-dependent guardband measurements with absolute errors of latest Raspberry Pi models B. Triangles indicate lower ($\bigtriangledown$) and upper ($\bigtriangleup$) frontier measurements while bullets ($\bullet$) indicate nominal measurements.

was successfully executed, the CPU core voltages are reduced in the configuration and the system is rebooted. These stages are repeated until either the system crashes due to too many faults or because the system can no longer boot as a result of insufficient supply voltage.

Figure 6.4 shows the safe, critical and failure voltage regions for a temperature-based guardband analysis using a multiplication benchmark that was executed in a single-threaded process by a billion loop iterations. The result of the multiplication is verified by the benchmark after each loop iteration. In the safe region depicted by the blue color no failures occurred during benchmark execution. Further undervolting into the critical region depicted by the yellow color resulted in occasionally failing processes. Undervolting into the failure region depicted by the red color rendered the system unusable. Nominal voltages depicted as connected black bullets are defined conservatively with a sufficient margin to the critical region. Never throughout the entire guardband analysis of the three Raspberry Pi models did the multiplication benchmark identify an incorrect product. It appears that the multiplication instruction is not on a timing critical path, even in the critical region, and faults are primarily injected in other processes.

Undervolting has a direct dependency with the temperature of the SoC as show by Figure 6.4. In particular with the Raspberry Pi 3B voltages of all regions are slightly rising with increasing temperatures. These voltage increments are due to the AVS system making adjustments based on the temperature of the SoC. The mechanics of the AVS system on the Raspberry Pi are unknown so far but the results of the temperature-based guardband analysis indicate that temperature is an input parameter. Resistivity of the circuitry increases with temperature which is the reason for the voltage adjustments by the AVS system. The Raspberry Pi 4B was missing overclocking support from the firmware[1] at the time of evaluation and could only be undervolted once to level -1 by $-15\,\mathrm{mV}$. Connected blue triangles indicate the undervolting limit for the Raspberry Pi 4B in Figure 6.4. Some basic overheating protection is observed in the range of $50\,^{\circ}\mathrm{C}$ to $70\,^{\circ}\mathrm{C}$ lowering nominal voltage by $-12.5\,\mathrm{mV}$. According to the temperature-based guardband analyses $-75\,\mathrm{mV}$ for the 3B, $-75\,\mathrm{mV}$ for the 3B+ and $-15\,\mathrm{mV}$ for the 4B are the ideal undervolting configurations within the safe region.

[1] https://www.raspberrypi.com/documentation/computers/config_txt.html#overclocking-options Last accessed 28.01.2022

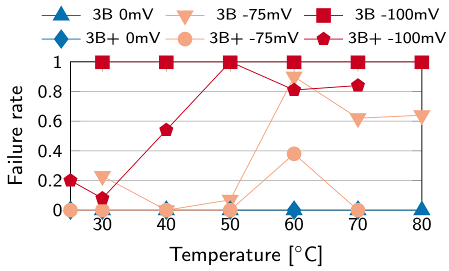


Figure 6.5: Temperature-dependent failure rate of Raspberry Pi model 3B and 3B+ at nominal and undervolted levels.

6.8.4 Implications on the Detection Method Success Rate

In order to produce the guardband analyses in Figure 6.4 741 measurements were run in total to determine safe parameters. Failures occurred in 265 of the 741 measurements that were either critical or crashed the system. The OS diagnoses the type of failure occurring in processes on the ARM architecture by consulting the Exception Syndrome Register [28]. An analysis of the 265 failed measurements identified 407 failed processes in which the following five types of failures occurred: (1) 46.4 % paging requests, (2) 26.7 % freeze during boot, (3) 20.3 % NULL pointer dereferences, (4) 5.4 % read from unreadable memory, and (5) 0.9 % write to read-only memory. All other types of failures, besides (2), generate a kernel oops, which are situations were the OS identified misbehaving processes and were execution of the system can possibility resumed. The diagnose of the kernel oops is written to the system log by the OS provided no error occurred in the meantime. Execution cannot be resumed after processes failures in case of missing dependencies or unavailable system resources. In particular when a kernel oops occurs in an interrupt handler the OS immediately raises panic and halts the system.

Figure 6.5 presents failure rates for the Raspberry Pi 3B and 3B+ models depending on the temperature for the 265 failed measurements. The Raspberry Pi 4B is not included in this figure because of the missing undervolting support in the firmware. Within the safe region and at nominal voltage no faults could be injected into the system during the measurements. At 60 °C the highest failure rates are observed on the 3B with 90 % and on the 3B+ with 40 %. Failure rates are shifting towards lower temperatures with more aggressive undervolting. In case of a -100 mV undervolting configuration, failures can reliably be injected at 40 °C on both models. At 70 °C failure rates are dropping on both models. Soft temperature throttling is the cause for the drop on the 3B+ but for the 3B no real reason could be identified. Among the 407 failed processes 33 unique processes were identified of which 34 % are user processes, 15 % kernel processes and 51 % unknown processes. The failed measurements do not contain a single instance of a failed multiplication benchmark.

6.8.5 Profit

Using benchmarks is an inexpensive and reliable option for evaluating the undervolt configuration of a system. `stress-ng` was the benchmark of choice since it provides a wide variety of stressors targeting specific OS functionalities. Surprisingly, not a single stressor failed during the execution of the benchmarks on the three models. A total of 169 stressors were executed sequentially with a runtime of 60 s. This is an affirmation for the cloud provider that the undervolt configuration determined in section 6.8.3 is sufficiently stable. These observations reconfirm what the state of the art had mentioned in section 6.4.2, that power viruses cannot be used for determining minimal voltage levels because there is no correlation to power consumption. The cloud provider has the erroneous belief that the undervolt configuration will keep the majority of cloud workloads within the safe region. Power viruses are not optimized for specific architectures and do not push components of processors to their limit. Additional functionalities implemented in power viruses generate sufficient overhead that prevents reaching these limits. Collecting statistics or verification of results are examples of such additional functionalities. Furthermore, these evaluations are usually not done under extreme conditions and only for a limited time because they are very time-consuming. Given these circumstances, this is an exploitable attack vector to push processors to their physical limits. The principle of the detection method is based on the idea to trip processes from the safe region into the critical region by raising temperatures through putting the processor under heavy load and consequently shifting regions limits. What makes the detection method particular is that there are no specific requirements for the program putting the processor under heavy load. Instead, the detection method relies on concurrent processes to fail because they happen to execute instructions that lie on the critical path.

The heat map in table 6.4 shows the ratio of ETRs for the undervolted and nominal configurations of the three Raspberry Pi models. Only the results of 27 of a total of 169 stressors are shown in the heat map. Measurements are divided into active and passive cooling setups. The ratios do not include energy consumed by the cooling solution because there is a wide range of cooling solutions which normally differ among cloud providers. Whether gains from passive cooling outweigh the costs of active cooling solutions is outside the scope of this evaluation and remains an interesting and open question. Higher energy efficiency is achieved in general on the 3B+ and 4B by actively cooling. The energy efficiency gains on the 4B are limited because of the missing undervolting support. Some stressors, such as hrtimers and judy, are 20 % respectively 7 % more energy-efficient with passive cooling on the 3B+. Examples of stressors that are more energy-efficient on the 4B with passive cooling are the futex stressor with 14 % and the sysfs stressor with 13 %. The 3B is an exception where 18 out of the 27 stressors are on average 4 % more energy-efficient using passive cooling. Occasionally, on the 3B+, it was even possible to successfully execute `stress-ng` benchmarks with undervolting set to −100 mV and active cooling. Average and maximum energy efficiency gains are summarized in table 6.5. `stress-ng` covers a wide spectrum of functionalities such that there is a large variance in the number of operations between stressors. Stressors with large numbers of operations are more energy-efficient since they do more operations in the same timeframe.

Table 6.4: STRESS-NG ETR heat map indicating the relative energy efficiency for an undervolted setup compared to a nominal setup. The darker the shade, the more energy-efficient the stressor ran.

Cooling	Model	Undervolt	aio	atomic	bsearch	clock	fork	futex	get	hrtimers	hsearch	icache	judy	kcmp	kill	lsearch	membarrier	mergesort	msg	pipe	poll	sem	sigsegv	sysfs	timer	tsearch	urandom	vm-rw	wcs
active	3B	−75 mV	0.94	0.95	0.96	0.95	0.92	1.02	1.03	0.90	0.95	0.95	0.99	0.93	0.93	0.94	1.02	0.94	0.91	0.96	0.95	0.93	0.94	0.91	0.94	0.99	0.95	0.96	0.94
	3B+	−75 mV	0.89	0.94	0.93	0.93	0.87	0.99	0.94	1.00	0.95	0.94	1.01	0.93	0.96	0.94	0.97	0.94	0.95	0.92	0.95	0.93	0.93	0.92	0.94	0.95	0.95	0.97	0.94
	4B	−15 mV	1.01	0.99	1.02	0.99	1.02	0.98	1.00	1.06	1.00	0.98	0.96	1.00	1.04	0.99	0.96	1.00	0.97	0.70	0.98	0.98	0.99	1.00	0.97	0.91	0.98	1.00	0.99
passive	3B	−75 mV	0.88	0.95	0.92	0.93	0.91	0.66	0.76	0.63	0.94	0.94	0.97	0.93	0.94	0.94	0.93	0.93	0.92	1.03	0.93	0.95	0.92	0.95	0.92	0.96	0.94	0.96	0.93
	3B+	−75 mV	0.95	0.95	0.96	0.95	0.98	0.97	0.99	0.80	0.95	0.95	0.94	0.95	0.95	0.95	0.98	0.95	0.95	0.96	0.94	0.95	0.94	0.91	0.95	0.97	0.96	0.97	0.95
	4B	−15 mV	1.00	0.97	1.02	0.99	1.01	0.84	1.00	1.10	0.99	1.03	0.99	0.98	1.00	1.00	0.97	1.00	1.03	1.01	1.00	1.05	1.04	0.87	1.00	0.91	0.99	0.97	0.99

Table 6.5: Summary of energy efficiency gains

Cooling	Model	Avg ETR	Max ETR stressor	Max ETR
	3B	5 %	hrtimers	10 %
active	3B+	6 %	fork	6 %
	4B	2 %	hrtimers	30 %
	3B	9 %	hrtimers	37 %
passive	3B+	6 %	hrtimers	20 %
	4B	1 %	futex	16 %

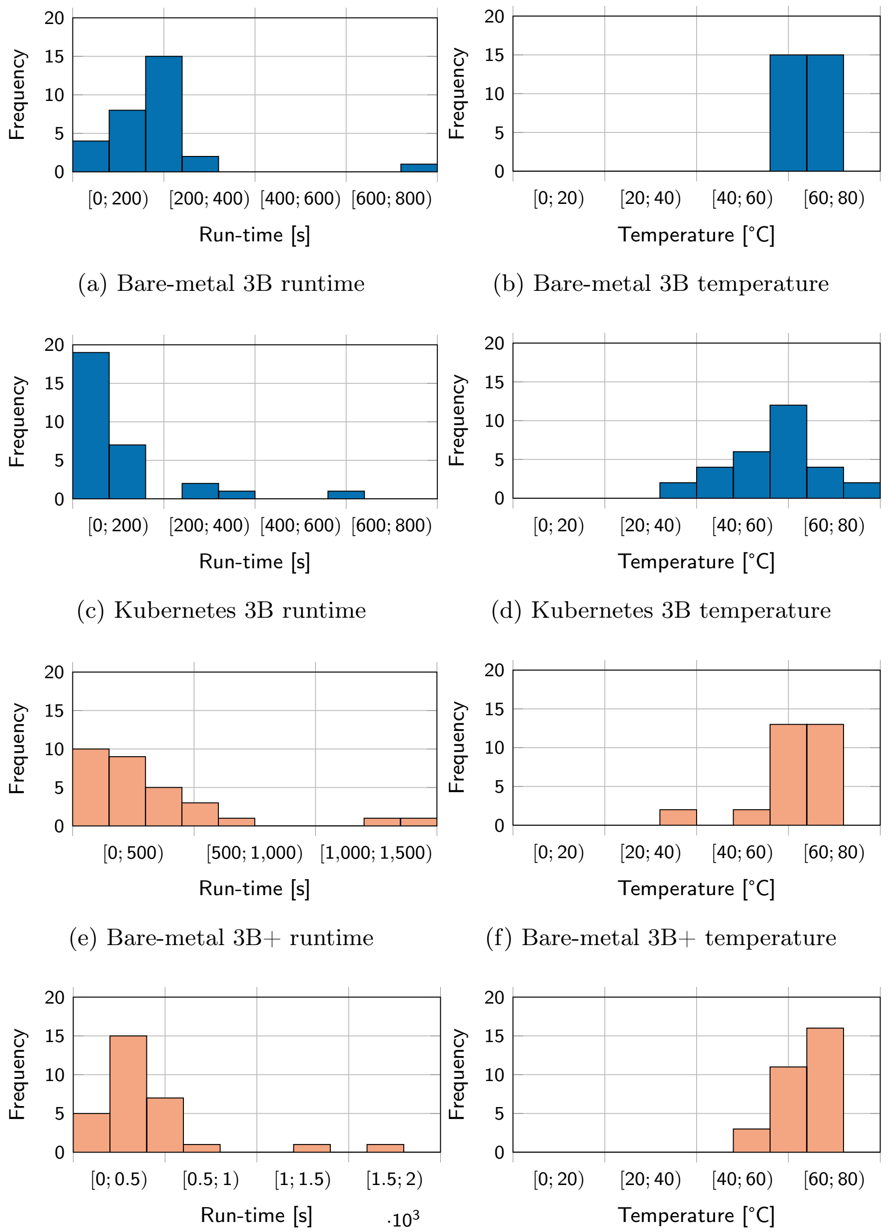
6.8.6 Detecting the Scrooge Attack

The scrooge attack is deployed by the cloud provider on passively cooled bare-metal and container instances on the 3B and the 3B+. Container instances were deployed on the Raspberry Pi using a small Kubernetes cluster. Virtual machines were not included in the evaluation because they are impracticable on the Raspberry Pi. The detection method is deployed by the cloud user over SSH connections to the bare-metal and container instances. Parameters relevant for the detection method are the runtime and the temperature. Cloud users can coordinate detection method runs if those two parameters were quantified, in order to simultaneously crash cloud instances.

The histograms in Figure 6.6 indicate the runtime and temperature buckets for 30 instance crashes. Temperatures of instance crashes are in line with the failure analysis from the temperature-based guardband analysis in section 6.8.4. Runtimes do not cluster as much and crashes are spread over a wider range. The runtime parameter is difficult to control and depends on the heat dissipation of the processor and the rate of the rising temperature. The histograms indicate a clear difference for the thermal designs of two models. Bare-metal instances were crashed on the 3B by the detection method on average after 175 s at 62 °C and container instances after 30 s at 62 °C. On the 3B+ bare-metal instances were crashed by the detection method on average after 145 s at 62 °C and container instances after 250 s at 62 °C. An interesting observation is that container instances crashed on the 3B earlier than bare-metal instances. Additional computing requirements for the container environment may explain this behavior.

6.9 Discussion

Undervolting improves the energy efficiency of processors in general and cloud providers can increase energy efficiency on average by 5 % and up to 37 % for specific workloads on ARM processors. By undervolting their own cloud infrastructure, cloud providers lower the margin of error but undergo major risks. Reliable detection of undervolted cloud instances by cloud users exploits the low margin of error through critical environments. The detection method relies on simple programs putting the processor under heavy load and maximizing power consumption in order to raise temperatures. Power or micro viruses are ideal candidates for such programs because they can be written using simple instructions that are less prone to timing violations under critical conditions. This is an important property which allows these programs to run until cloud instances become unavailable due to unrecoverable crashes. The detection method is



(a) Bare-metal 3B runtime

(b) Bare-metal 3B temperature

(c) Kubernetes 3B runtime

(d) Kubernetes 3B temperature

(e) Bare-metal 3B+ runtime

(f) Bare-metal 3B+ temperature

(g) Bare-metal 3B+ runtime

(h) Bare-metal 3B+ temperature

Figure 6.6: Runtime and temperature histograms of Raspberry Pi 3B and 3B+ bare-metal and container instances

scalable which helps adjust the failure injection rate and the crash time. Cloud providers can to some extend defuse the detection method by an active cooling system. The question that has not been addressed by this work is whether gains from undervolting the infrastructure outweigh costs for such cooling systems.

For cloud users the options to detect undervolted ARM instances remain very limited. Violation of the SLA through crashing instances by non-selective fault injection into other processes circumvents all security mechanisms installed by cloud providers hiding the undervolted state. Faults are injected by gradually pushing processors towards critical thermal conditions. These critical thermal conditions are reached by maximizing the load and power consumption of the processor in order to dissipate heat that raises temperatures. The temperature-based guardband analysis in section 6.8.4 clearly indicated increasing fault injection rates at higher temperatures. AVSs are designed to mitigate core aging and adjust platform parameters at runtime for optimal operational settings under various conditions. In case of the Raspberry Pi it remains unclear whether the adjustments made by the AVS are insufficient because they are specifically designed for nominal operation or that adjustments are limited to prevent thermal running. If the AVS had made larger adjustments, then it would not be possible to make use of the previously outlined detection method. The presented detection method is a proof of concept that requires further refinement in order to enable simultaneous crashes in multi-instance deployments and accurate prediction of crash times. Deterministic fault injection is a difficult objective to achieve as it depends on many factors the cloud user cannot influence. These factors are the instruction circuit design, process variation, executing processes and scheduling. Most of these factors are inherently random and cannot easily be influenced. The answers to the initially posed research questions in section 6.2 are as follows.

1. The cloud provider has many options to hide the undervolted state of the infrastructure. A mechanism must be installed that intercepts read requests from cloud users and substitutes undervolted values by nominal values. This mechanism can then be installed in the user space, kernel space or firmware depending of the cloud instance type.

2. The cloud user can detect undervolted instances without having to access the undervolt configuration. Faults are injected through power or micro viruses by reaching critical thermal conditions. Non-selective fault injection and imprecise control over crash time remain drawbacks.

Clouds can benefit from undervolting because of conservative voltage margins defined by processors manufacturers. Extreme undervolting close to the frontier of the critical region should not be the goal. As demonstrated by this chapter, the reliability of instructions and the stability of instances are at stake when operating near critical voltage margins. Even moderate undervolting resulting in minor power savings may accumulate to significant energy savings over the scale of the entire cloud infrastructure. In order to enable undervolting at scale, processors should include components such as ECC that guarantee reliable execution under critical conditions. ParaMedic and ParaDox mentioned in section 6.4.4 propose such architectures. The research interest in undervolting for energy savings or as side channel in attacks and voltage margins demonstrate that there are many more opportunities to explore.

The evaluation in section 6.8 is demonstrating how soft faults can be injected into cloud processes of IoT services and lead to incorrect results. It should be noticed that the presented Scrooge Attack is not limited to ARM processors or cloud instances and can be applied in similar ways to other processor architectures and platforms. This makes undervolting an universal technique that is generally applicable for improving the energy efficiency of processors. IoT services can apply undervolting in order to improve the overall energy efficiency of the service and in particular to extend the battery lifetime of IoT devices. However, undervolting needs to be applied carefully by selecting safe voltage margins that are specific to each device of the IoT service. Automated methods, such as the temperature-based guardband analysis, can be used for determining safe voltage margins.

7 Conclusion

7.1 Summary

IoT services comprise embedded, mobile, edge and cloud devices which are exposed to various threats from which they need to be protected. Securing IoT services through cryptographic primitives and establishing trust models are challenges developers have been confronted with. Energy efficiency and scalability are important properties to take into account when designing such services. These properties determine the lifetime of batteries for embedded and mobile devices as well as the availability and reliability of IoT services. The performance of entirely software-based solutions, such as homomorphic encryption, are not yet qualified for efficiently handling current workloads. In particular complex operations introduce overhead that is orders of magnitude beyond native operations. TEEs are promising hardware-based solutions providing the necessary confidentiality and integrity guarantees for enabling secure IoT services. TrustZone is evaluated for energy efficiency and scalability within this context of secure IoT services. The evaluation of the energy efficiency and scalability for TrustZone is done using three prototypes for secure IoT services. Various benchmarks as well as prototypes for shared memory, secure storage and network performance have been used for identifying compromises in the design of secure IoT services. Developing secure IoT services is exemplified on the FOC prototype and evaluated by executing confidential smart contracts in a small-scale deployment on Raspberry Pis.

IoT devices include embedded and mobile devices that are resource limited and battery powered. Data collected by these IoT devices can often not be stored or processed locally and has to be offloaded to resources available at the edge or in the cloud. Transmitting data is an energy intensive operation, especially if connections are wireless. Minimizing the data volume and transmission times saves network bandwidth and increases the uptime and battery lifetime of IoT devices. Fully integrated cloud IoT services provide network protocols which rely on file transfer or involve complex message exchanges. Hermes is a lightweight proof-of-concept network protocol that leverages GD as novel compression technique. The network protocol takes advantage of spacial and temporal correlations for reducing data transmission and improving energy efficiency on IoT devices. The network bandwidth for each link decreases as the data volume grows due to increasing matching rates from deduplication. Low network bandwidth requirements and high compression ratios make Hermes highly scalable. GD is computationally less demanding than common compression algorithms, such as LZW or DEFALTE, and specifically designed for resource limited devices.

Clouds have taken a central role in our lives and are managing our digital existence in the Internet. Everything we share digitally, be it documents, pictures, videos, messages or comments, is already or will eventually end up in the cloud. The cloud is in particular a key component for processing and storing data collected by IoT services. Energy efficiency is thereby becoming more and more important for cloud data centers by providing low power instances in order to reduce cooling costs. Conservative voltage margins of processors in cloud instances can be reduced through undervotling in order to improve energy efficiency. The energy gains from reducing voltage margins are increasing as returns of DVFS are diminishing by scaling down processors. While energy gains appear marginal for single instances, they scale with every instance and, due to their strongly increasing number, can reach substantial energy savings. Reliability of operations is at stake as supply voltages of processors are reduced to safe minimum voltages. Software induced changes to thermal conditions from power or micro viruses can move processors towards critical voltage margins. As a consequence, undervolted instances become susceptible to several attacks exploiting critical voltage margins.

7.2 Contributions

This thesis attempts to quantify the impact of TrustZone with OP-TEE as a TEE on energy efficiency, scalability, and security of IoT services. Various benchmarks and prototypes of secure IoT services demonstrate how restrictions of TEEs in combination with the design of TrustZone can negatively affect energy efficiency and scalability. Each TEE using TrustZone has its own restrictions and there is no universal solution that can be applied during development across all TEEs. For example, OP-TEE prioritizes security by minimizing the TCB through a TEE implementation with the necessary standard-conforming functionalities. On the one hand OP-TEE is enabling high throughput with small requests, whereas on the other hand TrustZone is limiting this throughput because of the many EL context switches requests have to traverse between CApps in the normal world and TAs in the secure world. The world switching overhead has been demonstrated using a secure storage prototype and exists also for network-enabled TAs. `iperfTZ` has been specifically developed as a network performance tool for identifying network bottlenecks for secure IoT services. With `iperfTZ` the world switching overhead has also been identified when OP-TEE is borrowing the REE network stack for transmitting packets. In order to improve energy efficiency and scalability of secure IoT services on TrustZone in general, frequent transitions in and out of TEEs should be avoided. This can be achieved by either buffering requests and transmitting them in a single bundle, if latency is not an issue, or by implementing trusted drivers and accessing hardware directly from the secure world.

Smart contracts are enabling new business relations but the risks of publicly exposing assets and business interactions are limiting the use of this DLT. The confidentiality and integrity of the smart contract, the data it is operating on and the transactions are in currently deployed blockchain systems not protected. Isolating the smart contract execution and shielding it by TEEs is providing the necessary confidentiality and integrity guarantees for realizing such business interactions. With FOC the modular architecture of Fabric is taken advantage of in order to isolate the execution of smart contracts in TrustZone for IoT devices. FOC is also proposing a heterogeneous approach involving different processor architectures for executing confidential smart contracts with Fabric. There are many challenges during the design and implementation of secure IoT services which are demonstrated at the example of FOC. It is in particular shown how limitations in TrustZone and OP-TEE are restricting the design of the prototype for confidential smart contracts.

Hermes is a protocol for IoT that specifically addresses challenges of energy efficiency and network bandwidth. GD is the technology behind Hermes enabling energy efficiency through high compression ratios. Hermes is based on a hop-by-hop scheme with a simple message exchange between nodes in order to minimize network bandwidth, in particular for resource limited embedded and mobile devices using wireless connections. Caching capabilities of IoT and edge devices is key for reducing network bandwidth at the cloud and shifting traffic to the edge. Results indicate that GD can save several orders of magnitude in network bandwidth compared to raw data transmission and data deduplication. Additional energy spend compressing and deduplicating data with GD is compensated by reduced transmission times. The energy consumption of GD can in some instances even be lower than data deduplication.

The Scrooge Attack is a novel scenario in which a powerful and malicious cloud provider purposely undervolts the cloud infrastructure for profit. So far, the state of the art embraces undervolting as a technique for reducing voltage margins to safe minimal voltages in order to save energy. Undervolting is an universally applicable technique that improves the energy efficiency of processors and can extend the battery lifetime of IoT devices as well as improving the availability of IoT services. The temperature-based guardband analysis has been developed as a mechanism for automatically selecting appropriate undervolting configurations and assisting in the search of safe minimal voltages. With this mechanism the reliability of undervolted cloud instances under critical conditions has been investigated. According to the result of this investigation a detection method based on power or micro viruses was developed, which allows cloud users to remotely identify undervolted cloud instances by violation of the SLA in the presence of a strong threat model trying to prevent the identification. Cloud users can reliably crash cloud instances using the detection method independent of any countermeasures by cloud providers. The attack, the detection method and the temperature-based guardband analysis have been demonstrated on the ARM architecture but are not limited to it. It is expected that the attack, the detection method and the temperature-based guardband analysis are applicable to any other architecture.

7.3 Implications

Secure services can be implemented in energy-efficient and scalable ways while adhering to restrictions of TEEs. The choice of a TEE is only partially dependent on energy efficiency and scalability while primary decision-making factors are target platforms and security guarantees. Many vulnerabilities have been discovered in hardware-enforced isolation technologies such as TrustZone [42, 70, 75, 178, 259, 267, 269, 288, 313], SGX [57, 68, 71, 91, 126, 228, 302] and SEV [55, 56, 98, 139, 140, 182, 222, 223, 224, 261, 307]. Open questions are whether services are more secure using TEEs and if it is worth trading off energy efficiency and scalability for it. TEEs depend on software which may contain bugs and errors or does not cover certain use cases. These flaws can be exploited in some way by malicious users to gain access to confidential and sensitive information. There is never going to be a perfectly secure system independent of the security guarantees and provided security measures. Over time, methods will be developed to circumvent the security of any system. TEEs are instruments to raise the hurdles for circumventing security guarantees. Services using TEEs can be more secure than other isolation techniques such as virtualization and containers, and it is worth trading off energy efficiency and scalability for it. As long as TEE limitations are respected, there are only minor drawbacks in terms of energy efficiency and scalability, and the enhanced security guarantees predominate.

Hermes as a proof-of-concept protocol has demonstrated the potential of GD applied to IoT and how this technique can enable energy efficiency and scalability. There is the possibility that GD can be integrated in some alternate form into established protocols such as AMQP, MQTT or CoAP. CDNs and caching services are further domains where Hermes could be useful. In these use cases the roles of source and sink nodes are reversed and data is moved from servers through the edge to mobile and client devices. Since caching systems provide storage, bases would be stored locally on nodes along the route before being forwarded to clients. However, because roles are reversed and client nodes become sink nodes, knowledge bases are much more limited than server or cloud storage. Consequently, matching rates for bases are lower and less network bandwidth can be saved. Instead of considering a hierarchical network topology with a single sink node as root, more complex network topologies need to be integrated into the protocol. For example, intermediate nodes could have links to multiple clients and broaden their knowledge base in order to increase matching rates and reduce network bandwidth. It is not excluded that the concepts implemented in Hermes can be applied to other domains that have not been considered until now.

Operating cloud instances at reduced voltage margins is a risky endeavor for cloud providers. Cloud providers need to evaluate the gains in energy efficiency against the loss in reliability of cloud instances. The state of the art is embracing undervolting in the cloud but the current work has demonstrated that not even safe minimum voltages are sufficiently reliable under critical conditions. For undervolting to prevail, microarchitectural changes need to be made to processors in order to include ECCs and other mechanisms that improve the reliability when operating at reduced voltage margins. The Scrooge Attack did not attempt to exploit cloud instances in any other way than for crashes. Related work, however, has demonstrated how cryptographic primitives and even TEEs can be successfully undermined using undervolting. It is expected that such attacks are to some extend applicable when operating at reduced voltage margins.

7.4 Future Work

7.4.1 Energy Efficiency Under Realistic Conditions

The evaluation of secure IoT services on TrustZone using OP-TEE have clearly demonstrated the direct influence of TEE restrictions on energy efficiency, scalability and security. Energy efficiency has been measured under idealized conditions which are not necessarily representative for IoT deployments. For example sensor nodes are taking measurements and aggregating them at regular intervals before transmitting the data and going back into deep idle states. The CPU of sensor nodes have bursty behavior with varying throughput that is not covered by any energy efficiency unit used in the current work. Microprocessor ETR (METR) [58] extends ETR as an energy efficiency unit by taking into account the idle state of processors. The challenges with METR are accurately measuring CPU bursts, multiple power states and multi-core CPUs. In order to measure the power consumption of CPU bursts power meters need to have sufficient resolution for determining the ratio between performance and idle states. There is a range of possible OPPs in between the highest and lowest states that CPUs can use which are not considered by METR. Furthermore, each CPU core can operate at different OPPs such that highest and lowest states are rarely attained. Determining realistic energy efficiency numbers remains an open issue.

7.4.2 Confidential Smart Contracts

The FOC prototype does not provide remote attestation as a mechanism for users to verify that smart contracts are being executed on genuine TrustZone-enabled processors. Implementing remote attestation involves accessing the HUK of the processor for deriving new keys. These HUKs, however, are kept confidential by most manufacturers and require NDAs in order to disclose the access. The issue can be circumvented by lowering security guarantees and installing HUKs in software as it can done in OP-TEE. Another challenge to consider is the replication of chaincodes in FOC. The prototype is using pre-installed chaincodes which avoids the need to replicate them across the Fabric network. A simple solution would use the same build key with OP-TEE on all nodes in the network. The security of the entire blockchain framework for such a solution would then depend on a single key, which can compromise the entire system if the key is leaked at any point in time.

7.4.3 Impact of Different ECCs and Pre-Processing

The Hermes protocol is not taking full advantage of GD. Support for other ECCs than Hamming codes, such as Reed-Solomon, or pre-processing, such as delta encoding and offset removal, are not implemented. Analyzing and comparing the energy efficiency and scalability of the protocol with support for multiple error corrections provides new insight about compression ratios for IoT data. What is the impact of larger Hamming distances and are there optimal Hamming distances for IoT data in terms of energy efficiency, scalability and network bandwidth? These are further questions that could be addressed for Hermes.

7.4.4 Detection Method for Multi-Instance Deployments

The Scrooge Attack is in a proof-of-concept state and can be expanded in several ways. One way that is to be explored in a future work are synchronizing crashes across multiple instances. The detection method is in its current state applicable to single cloud instances. Deployments including multiple cloud instances cannot make use of the detection method, unless it is possible to implement simultaneous cloud instance crashes. SLAs are formulated such that cloud users are only covered if this requirement is satisfied. The challenge of expanding the detection method by simultaneous crashes lies in the unpredictability of process variation. It is possible that the same type of cloud instance behaves differently depending on the process variation. The second way is to explore feasible exploits that take advantage of critical conditions in cloud instances. Side-channel attacks making use of undervolting are potential starting points.

Appendices

Publications

1. C. Göttel, L. Nielsen, N. Yazdani, P. Felber, D.E. Lucani, V. Schiavoni, "Energy-Efficient IoT Networks with Hermes: A Comparison," unpublished.

2. J. Ménétrey, C. Göttel, A. Khurshid, Marcelo Pasin, Pascal Felber, Valerio Schiavoni, S. Raza, "Attestation Mechanisms for Trusted Execution Environments Demystified", under submission.

3. J. Ménétrey, C. Göttel, Marcelo Pasin, Pascal Felber, Valerio Schiavoni, "An Exploratory Study of Attestation Mechanisms for Trusted Execution Environments," 5th Workshop on System Software for Trusted Execution (SysTEX), 2022, URL: https://systex22.github.io/papers/systex22-final79.pdf

4. C. Göttel, K. Parasyris, O. Unsal, P. Felber, M. Pasin and V. Schiavoni, "Scrooge Attack: Undervolting ARM Processors for Profit: Practical experience report," 2021 40th International Symposium on Reliable Distributed Systems (SRDS), 2021, pp. 187-197, doi: 10.1109/SRDS53918.2021.00027

5. B. Salami, K. Parasyris, A. Cristal, O. Unsal, X. Martorell, P. Carpenter, R. De La Cruz, L. Bautista, D. Jimenez, C. Alvarez, S. Nabavi, S. Madonar, M. Pericàs, P. Trancoso, M. Abduljabbar, J. Chen, P.N. Soomro, M. Manivannan, M. vor dem Berge, S. Krupop, F. Klawonn, A. Al-Mekhlafi, S. May, T. Becker, G. Gaydadjiev, H. Salomonsson, D. Dubhashi, O. Port, Y. Etsion, D. Le Quoc, C. Fetzer, M. Kaiser, N. Kucza, J. Hagemeyer, R. Griessl, L. Tigges, K. Mika, A. Hüffmeier, M. Pasin, V. Schiavoni, I. Rocha, C. Göttel, P. Felber, "LEGaTO: Low-Energy, Secure, and Resilient Toolset for Heterogeneous Computing," 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2020, pp. 169-174, doi: 10.23919/DATE48585.2020.9116362

6. C. Müller, M. Brandenburger, C. Cachin, P. Felber, C. Göttel, V. Schiavoni, "TZ4Fabric: Executing Smart Contracts with ARM TrustZone : (Practical Experience Report)," 2020 International Symposium on Reliable Distributed Systems (SRDS), 2020, pp. 31-40, doi: 10.1109/SRDS51746.2020.00011

7. C. Göttel, L. Nielsen, N. Yazdani, P. Felber, D.E. Lucani, V. Schiavoni, "Hermes: Enabling Energy-Efficient IoT Networks with Generalized Deduplication," Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems (DEBS '20), Association for Computing Machinery, New York, NY, USA, 2020, 133-136, doi: 10.1145/3401025.3404098

8. C. Göttel, P. Felber, V. Schiavoni (2019) iperfTZ: Understanding Network Bottlenecks for TrustZone-Based Trusted Applications. In: M. Ghaffari, M. Nesterenko, S. Tixeuil, S. Tucci, Y. Yamauchi (eds) Stabilization, Safety, and Security of Distributed Systems. SSS 2019. Lecture Notes in

Computer Science, vol 11914. Springer, Cham. 10.1007/978-3-030-34992-9_15

9. C. Göttel, P. Felber, V. Schiavoni (2019) Developing Secure Services for IoT with OP-TEE: A First Look at Performance and Usability. In: J. Pereira, L. Ricci (eds) Distributed Applications and Interoperable Systems. DAIS 2019. Lecture Notes in Computer Science, vol 11534. Springer, Cham. 10.1007/978-3-030-22496-7_11

10. I. Rocha, C. Göttel, P. Felber, M. Pasin, R. Rouvoy, V. Schiavoni, "Heats: Heterogeneity- and Energy-Aware Task-Based Scheduling," 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2019, pp. 400-405, doi: 10.1109/EMPDP.2019.8671554.

11. C. Göttel, R. Pires, I. Rocha, S. Vaucher, P. Felber, M. Pasin, V. Schiavoni, "Security, Performance and Energy Implications of Hardware-Assisted Memory Protection Mechanisms on Event-Based Streaming Systems," 2018 IEEE 37th International Symposium on Reliable Distributed Systems (SRDS), 2018, pp. 264-266, doi: 10.1109/SRDS.2018.00042.

12. C. Göttel, R. Pires, I. Rocha, S. Vaucher, P. Felber, M. Pasin, V. Schiavoni, "Security, Performance and Energy Trade-Offs of Hardware-Assisted Memory Protection Mechanisms," 2018 IEEE 37th International Symposium on Reliable Distributed Systems (SRDS), 2018, pp. 133-142, doi: 10.1109/SRDS.2018.00024.

13. A. Cristal, O.S. Unsal, X. Martorell, P. Carpenter, R. De La Cruz, L. Bautista, D. Jimenez, C. Alvarez, B. Salami, S. Madonar, M. Pericàs, P. Trancoso, M. vor dem Berge, G. Billung-Meyer, S. Krupop, W. Christmann, F. Klawonn, A. Al-Mekhlafi, T. Becker, G. Gaydadjiev, H. Salomonsson, D. Dubhashi, O. Port, E. Hadar, Y. Etsion, C. Fetzer, J. Hagemeyer, T. Jungeblut, N. Kucza, M. Kaiser, M. Porrmann, M. Pasin, V. Schiavoni, I. Rocha, C. Göttel, P. Felber, "LEGaTO: first steps towards energy-efficient toolset for heterogeneous computing," Proceedings of the 18th International Conference on Embedded Computing Systems: Architectures, Modeling, and Simulation (SAMOS '18), Association for Computing Macinery, New York, NY, USA, 2018, doi: 10.1145/3229631.3239370

14. A. Cristal, O.S. Unsal, X. Martorell, P. Carpenter, R. De La Cruz, L. Bautista, D. Jimenez, C. Alvarez, B. Salami, S. Madonar, M. Pericàs, P. Trancoso, M. vor dem Berge, G. Billung-Meyer, S. Krupop, W. Christmann, F. Klawonn, A. Al-Mekhlafi, T. Becker, G. Gaydadjiev, H. Salomonsson, D. Dubhashi, O. Port, Y. Etsion, V. Nowack, C. Fetzer, J. Hagemeyer, T. Jungeblut, N. Kucza, M. Kaiser, M. Porrmann, M. Pasin, V. Schiavoni, I. Rocha, C. Göttel, P. Felber, "LEGaTO: towards energy-efficient, secure, fault-tolerant toolset for heterogeneous computing," Proceedings of the 15th ACM International Conference on Computing Frontiers (CF '18). Association for Computing Machinery, New York, NY, USA, 2018, doi: 10.1145/3203217.3205339

Bibliography

[1] AWS Graviton Processor. https://aws.amazon.com/ec2/graviton/. Accessed on 28.01.2022.

[2] AWS Nitro System. https://aws.amazon.com/ec2/nitro/. Accessed on 28.01.2022.

[3] Neoverse N1. https://developer.arm.com/ip-products/processors/neoverse/neoverse-n1. Accessed on 28.01.0222.

[4] Filesystem in Userspace. https://github.com/libfuse/libfuse, 2001. Accessed on 28.01.2022.

[5] Expect. https://core.tcl-lang.org/expect/index, August 2009. Accessed on 28.01.2022.

[6] stress-ng. https://github.com/ColinIanKing/stress-ng, March 2014. Accessed on 28.01.2022.

[7] OpenDedup File System. https://opendedup.org/odd/2016/09/15/filesystem/, September 2016. Accessed on 28.01.2022.

[8] Ampere Altra: The World's First Cloud Native Processor. https://amperecomputing.com/altra/, November 2020. Accessed on 28.01.2022.

[9] ISO/IEC JTC 1. *ISO/IEC 11889:2015*. International Organization for Standardization, 2 edition, August 2015.

[10] Bernard Abodba. *The Online User's Encyclopedia*, chapter How the Internet Came to Be. Addison-Wesley, 1993. Cerf, Vinton.

[11] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, Alex J. Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. Imperfect Forward Secrecy: How Diffie-Hellmann Fails in Practice. In *22nd ACM Conference on Computer and Communications Security*, October 2015.

[12] Advanced Micro Devices, Inc. AMD EPYC Datacenter Processor Launches with Record-Setting Performance, Optimized Platforms, and Global Server Ecosystem Support. https://www.amd.com/en/press-releases/amd-epyc-datacenter-2017jun20, June 2017. Accessed on 28.01.2022.

[13] Advanced Micro Devices, Inc. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf, January 2020. Accessed on 28.01.2022.

[14] Advanced Micro Devices, Inc. *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, 3.38 edition, 2021.

[15] Defense Advanced Research Projects Agency. Internet Protocol. RFC791, September 1981.

[16] Sam Ainsworth and Timothy M. Jones. Parallel Error Detection Using Heterogeneous Cores. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 338–349, 2018.

[17] Sam Ainsworth and Timothy M. Jones. ParaMedic: Heterogeneous Parallel Error Correction. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 201–213, 2019.

[18] Sam Ainsworth, Lionel Zoubritzky, Alan Mycroft, and Timothy M. Jones. ParaDox: Eliminating Voltage Margins via Heterogeneous Fault Tolerance. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 520–532, 2021.

[19] Alciom. *PowerSpy2 User Manual*, 01.01 edition, March 2013. AL/RL/1236/004.

[20] Alibaba Cloud. Install SGX. `https://www.alibabacloud.com/help/doc-detail/108507.htm`, September 2020. Accessed on 28.01.2022.

[21] Tiago Alves and Don Felton. TrustZone: Integrated Hardware and Software Security. *Information Quarterly*, 3(4):18–24, 2004.

[22] Gene M. Amdahl. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. Association for Computing Machinery.

[23] Ampere Computing. Ampere eMAG 8180 64-bit Arm Processor. Technical Report Amp 2018-0007, 4655 Great America Parkway, Suite 601, Santa Clara, CA 05054, 2018.

[24] Ittai Anati, Shay Gueron, Simon P. Johnson, and Vincent R. Scarlata. Innovative Technology for CPU Based Attestation and Sealing. `https://www.intel.com/content/dam/develop/external/us/en/documents/hasp-2013-innovative-technology-for-attestation-and-sealing.pdf`, June 2013. Presented at HASP '13.

[25] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, pages 1–15, New York, NY, USA, 2018. Association for Computing Machinery. Article No. 30, 10.1145/3190508.3190538.

[26] Daniel Andzakovic. Extracting BitLocker Keys From a TPM. https://pulsesecurity.co.nz/articles/TPM-sniffing, March 2019. Accessed on 28.01.2022.

[27] Apache Software Foundation. https://github.com/apache/couchdb, 2005. Accessed on 28.01.2022.

[28] ARM Limited. *ARM Architecture Reference Manual*, i edition, July 2005. ARM DDI 0100I.

[29] ARM Limited. *ARM Security Technology*, 2009. PRD29-GENC-009492C.

[30] ARM Limited. *Arm Cortex-A72 MPCore Processor Technical Reference Manual*, r0p3 edition, December 2016. ARM 100095_0003_06_en.

[31] ARM Limited. *Isolation using virtualization in the Secure world*, 2018. LES-PRE-20349.

[32] ARM Limited. *Arm Neoverse N1 Core Technical Reference Manual*, r4p0 edition, April 2019. 100616_0400_00_en.

[33] ARM Limited. *Armv8-A virtualization*, 2019. Doc ID 102142.

[34] ARM Limited. *Introduction to AMBA AXI4*, 01 edition, August 2020.

[35] ARM Limited. *AMBA AXI and ACE Protocol Specification*, h.c edition, January 2021.

[36] Eric M. Aupperle. Merit - Who, What, and Why. *Library Hi Tech*, 16(1), 1998.

[37] Anys Bacha and Radu Teodorescu. Dynamic Reduction of Voltage Margins by Leveraging On-Chip ECC in Itanium II Processors. *SIGARCH Comput. Archit. News*, 41(3):297–307, June 2013.

[38] Leemon Baird. The Swirlds Hashgraph Consensus Algorithm: Fair, Fast, Byzantine Fault Tolerance. Technical Report SWIRLDS-TR-2016-01, Swirlds, May 2016.

[39] R. Barnes, M. Thomson, A. Pironti, and A. Langley. Deprecating Secure Sockets Layer Version 3.0. RFC7568, June 2015.

[40] J. Barr. Coming Soon - Graviton2-Powered General Purpose, Compute-Optimized, & Memory-Optimized EC2 Instances. https://aws.amazon.com/blogs/aws/coming-soon-graviton2-powered-general-purpose-compute-optimized-memory-optimized-ec2-instances/, December 2019. Accessed on 28.01.2022.

[41] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding Applications from an Untrusted Cloud with Haven. In *11th USENIX Symposium on Operating System Design and Implementation (OSDI 14)*, pages 267–283, Broomfield, CO, October 2014. USENIX Association.

[42] E. M. Benhani, L. Bossuet, and A. Aubert. The Security of ARM TrustZone in a FPGA-Based SoC. *IEEE Transactions on Computers*, 68(8):1238–1248, 2019.

[43] Tim Berners-Lee. TCP/IP based server for HyperText. https://www.w3.org/Daemon/old/V0.1/daemon.c, October 1990. Accessed on 28.01.2022.

[44] Daniel J. Bernstein. ChaCha, a variant of Salsa20. https://cr.yp.to/chacha/chacha-20080128.pdf, January 2008. Accessed on 28.01.2022.

[45] Benjamin Beurdouche, Karthikeyan Bhargavan, ANtoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. A Messy State of the Union: Taming the Composite State Machines of TLS. In *2015 IEEE Symposium on Security and Privacy*, pages 535–552, 2015.

[46] Burton H. Bloom. Space/Time Trade-Offs in Hash Coding with Allowable Errors. *Commun. ACM*, 13(7):422–426, July 1970.

[47] Mark Bohr. A 30 year retrospective on dennard's mosfet scaling paper. *IEEE Solid-State Circuits Society Newsletter*, 12(1):11–13, 2007.

[48] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. Association for Computing Machinery.

[49] Greg Boss, Padma Malladi, Dennis Quan, and Linda Legregni. Cloud Computing, October 2007.

[50] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[51] Marcus Brandenburger, Christian Cachin, Rüdiger Kapitza, and Alessandro Sorniotti. Trusted Computing Meets Blockchain: Rollback Attacks and a Solution for Hyperledger Fabric. In *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, pages 324–333, 2019.

[52] Marcus Brandenburger, Christian Cachin, Rüdiger Kapitza, and Alessandro Sorniotti. Trusted Computing Meets Blockchain: Rollback Attacks and a Solution for Hyperledger Fabric. In *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, pages 324–333, 2019.

[53] Broadcom Corporation. *VideoCore IV 3D Architecture Reference Guide*, videocoreiv-ag100-r edition, November 2013.

[54] Dominik Brodowski. Linux CPUFreq - CPU frequency and voltage scaling code in the Linux kernel. https://www.kernel.org/doc/html/latest/cpu-freq/index.html. Accessed on 28.01.2022.

[55] Robert Buhren, Hans Niklas Jacob, Thilo Krachenfels, and Seifert Jean-Pierre. One Glitch to Rule Them All: Fault Injection Attacks Against AMD's Secure Encrypted Virtualization. In *arXiv*, pages 1–14, 2021.

[56] Robert Buhren, Christian Werling, and Jean-Pierre Seifert. Insecure Until Proven Updated: Analyzing AMD SEV's Remote Attestation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, pages 1087–1099, New York, NY, USA, 2019. Association for Computing Machinery.

[57] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 991–1008, Baltimore, MD, August 2018. USENIX Association.

[58] Thomas D. Burd and Robert W. Brodersen. Energy Efficient CMOS Microprocessor Design. In *Proceedings of the Twenty-Eighth Annual Hawaii International Conference on System Sciences*, volume 1, pages 288–297, 1995.

[59] Vitalik Buterin. Ethereum: A Next-Generation Generalized Smart Contract and Decentralized Application Platform. `http://ethereum.org/ethereum.html`, January 2014. Accessed on 11.01.2014.

[60] Christian Cachin and Marko Vukolić. Blockchain Consensus Protocols in the Wild. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing (DISC 2017)*, volume 91 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. 10.4230/LIPIcs.DISC.2017.1.

[61] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pages 173–186, Pittsburgh, PA, USA, February 1999. USENIX Association.

[62] Stefania Cavallar, Bruce Dodson, Arjen K. Lenstra, Walter Lioen, Peter L. Montgomery, Brian Murphy, Herman te RIele, Karen Aardal, Jeff Gilchrist, Gérard Guillerm, Paul Leyland, Jöel Marchand, Fançois Morain, Alec Muffett, Chris Putnam, Craig Putnam, and Paul Zimmermann. Factorization of a 512-Bit RSA Modulus. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, pages 1–18, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[63] Cavium. ThunderX Family of Workload Optimized Processors. Technical report, 2315 N. First Street, San Jose, CA 95131, 2016.

[64] Vinton Cerf, Yogen Dalal, and Carl Sunshine. Specification of Internet Transmission Control Program. RFC675, December 1974.

[65] Anantha P. Chandrakasan, Samuel Sheng, and Robert W. Brodersen. Low-Power CMOS Digital Design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, 1992.

[66] D.L. Chaum. Computer Systems. Technical Report UCB/ERL M79/10, EECS Department, University of California, Berkeley, February 1979. `https://www2.eecs.berkeley.edu/Pubs/TechRpts/1979/28419.html`.

[67] D.L. Chaum. *Computer Systems Established, Maintained, and Trusted by Mutually Suspicious Groups*. PhD thesis, UC Berkeley, June 1982.

[68] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 142–157, 2019.

[69] Xiaoxin Chen, Tal Garfinkel, E. Christopher Lewis, Pratap Subrahmanyam, Carl A. Waldspurger, Dan Boneh, Jeffrey Dwoskin, and Dan R.K. Ports. Overshadow: A Virtualization-Based Approach to Retrofitting Protection in Commodity Operating Systems. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, pages 2–13, New York, NY, USA, 2008. Association for Computing Machinery.

[70] Yue Chen, Yulong Zhang, Zhi Wang, and Tao Wei. Downgrade Attack on TrustZone, 2017.

[71] Zitai Chen, Georgios Vasilakis, Kit Murdock, Edward Dean, David Oswald, and Flavio D. Garcia. VoltPillager: Hardware-based fault injection attacks against Intel SGX Enclaves using the SVID voltage scaling interface. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 699–716. USENIX Association, August 2021.

[72] Jie Cheng, Hongbo Jiang, Xiaoqiang Ma, Lanchao Liu, Lijun Qian, Chen Tian, and Wenyu Liu. Efficient Data Collection with Sampling in WSNs: Making Use of Matrix Completion Techniques. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pages 1–5, 2010.

[73] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 185–200, 2019.

[74] Siddhartha Chhabra, Brian Rogers, Yan Solihin, and Milos Prvulovic. SecureME: A Hardware-Software Approach to Full System Security. In *Proceedings of the International Conference on Supercomputing*, ICS '11, pages 108–119, New York, NY, USA, 2011. Association for Computing Machinery.

[75] Haehyun Cho, Penghui Zhang, Donguk Kim, Jinbum Park, Choong-Hoon Lee, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn. Prime+Count: Novel Cross-World Covert Channels on ARM TrustZone. In *Proceedings of the 34th Annual Computer Security Applications Conference*, ACSAC '18, pages 441–452, New York, NY, USA, 2018. Association for Computing Machinery.

[76] Sunoh Choi, Gabriel Ghinita, and Bertino Elisa. A Privacy-Enhancing Content-Based Publish/Subscribe System Using Scalar Product Preserving Transformations. In Pablo García Bringas, Abdelkader Hameurlain, and Gerald Quirchmayr, editors, *Database and Expert Systems Applications*, pages 368–384, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[77] Alonzo Church. An unsolvable problem of elementary number theory. *American journal of mathematics*, 58:345–363, 1936.

[78] Cisco Systems. Cisco Visual Networking Index: Forecast and trends, 2017–2022. White Paper, 2018.

[79] Cisco Systems. Cisco Annual Internet Report (2018–2023). White Paper, March 2020.

[80] Maxime Colmant and Guillaume Fieni. powerspy.scala. `https://github.com/Spirals-Team/powerspy.scala`, February 2018. Accessed on 28.01.2022.

[81] Computer Emergency Respomse Team (CERT), Carnegie Mellon University, Pittsburgh, PA. CA-1996-21: TCP SYN Flooding and IP Spoofing Attacks, September 1996.

[82] Core Specification Working Group. *Bluetooth Core Specification*, v5.2 edition, December 2019.

[83] Intel Corporation. Intel Software Guard Extensions for Linux. `https://01.org/intel-software-guard-extensions`. Accessed on 28.01.2022.

[84] Intel Corporation. Intel Tofino. `https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html`. Accessed on 28.01.2022.

[85] Intel Corporation. Intel Unveils New Developer Tools, Future Technologies Spanning Tablets, Analytics, Wearable Devices and PCs at Developer Forum. `https://newsroom.intel.com/news-releases/intel-unveils-new-developer-tools-future-technologies-spanning-tablets-analytics-wearable-devices-and-pcs-at-developer-forum/#gs.eags2a`, September 2014. Accessed on 28.01.2022.

[86] Intel Corporation. Chip Shot: Intel Unleashes Next-Gen Enthusiast Desktop PC Platform at Gamescom. `https://newsroom.intel.com/chip-shots/chip-shot-intel-unleashes-next-gen-enthusiast-desktop-pc-platform-at-gamescom/#gs.eahokw`, August 2015. Accessed on 28.01.2022.

[87] Intel Corporation. Lakefield: Hybrid CPU with Foveros Technology. `https://newsroom.intel.com/press-kits/lakefield/`, June 2020. Accessed on 28.01.2022.

[88] Intel Corporation. XuCode: An Innovative Technology for Implementing Complex Instruction Flows. `https://software.intel.com/content/www/us/en/develop/articles/software-security-guidance/secure-coding/xucode-implementing-complex-instruction-flows.html`, May 2021. Accessed on 28.01.2022.

[89] Microsoft Corporation. Open Enclave SDK. `https://openenclave.io/sdk`. Accessed on 28.01.2022.

[90] Microsoft Corporation. Open Enclave SDK - GitHub. `https://github.com/openenclave/openenclave`. Accessed on 28.01.2022.

[91] Jinhua Cui, Yu Zhijingcheng, Shweta Shinde, Prateek Saxena, and Zhiping Cai. SmashEx: Smashing SGX Enclaves Using Exceptions. In *28th ACM Conference on Computer and Communications Security (CCS)*, November 2021.

[92] Joan Daemen and Vincent Rijmen. The Rijndael Block Cipher. AES Proposal, September 1999.

[93] D.W. Davies, K.A. Bartlett, R.A. Scantlebury, and P.T. Wilkinson. A digital communications network for computers giving rapid response at remote terminals. In *Proceedings of the First ACM Symposium on Operating System Principles*, SOSP'67, pages 2.1–2.17, New York, NY, USA, 1967. Association for Computing Machinery.

[94] R.H. Dennard, F.H. Gaensslen, Hwa-Nien Yu, V.L. Rideout, E. Bassous, and A.R. LeBlanc. Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[95] Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in Key Dristirbution Protocols. *Commun. ACM*, 24(8):533–536, August 1981.

[96] P. Deutsch. DEFLATE Compressed Data Format Specification version 1.3. RFC1951, May 1996.

[97] T. Dierks and E. Roscorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC5246, August 2008.

[98] Zhao-Hui Du, Zhiwei Ying, Zhenke Ma, and Yufei Mai. Secure Encrypted Virtualization is Unsecure! `https://export.arxiv.org/pdf/1712.05090`, 2017. Accessed on 28.01.2022.

[99] D.L. Eager, J. Zahorjan, and E.D. Lazowska. Speedup versus efficiency in parallel systems. *IEEE Transactions on Computers*, 38(3):408–423, 1989.

[100] A.W. Eckford and Wei Yu. Density evolution for the simultaneous decoding of LDPC-based slepian-wolf source codes. In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.*, pages 1401–1405, 2005.

[101] Ecma International. *Near Field Communication - Interface and Protocol (NFCIP-1)*, thrid edition, June 2013.

[102] Ecma International. *The JSON data interchange syntax*, first edition, October 2013.

[103] Ahmed Elzanaty, Andrea Giorgetti, and Marco Chiani. Lossy Compression of Noisy Sparse Sources Based on Syndrome Encoding. *IEEE Transactions on Communications*, 67(10):7073–7087, 2019.

[104] D. Ernst, Nam Sung Kim, S. Das, S. Pant, R. Rao, Toan Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: a low-power pipeline based on circuit-level timing speculation. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, pages 7–18, 2003.

[105] Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, pages 365–376, 2011.

[106] ESnet / Lawrence Berkeley National Laboratory. iperf3. `https://software.es.net/iperf/`. Accessed on 28.01.2022.

[107] E.W. Felten. Understanding trusted computing: will its benefits outweigh its drawbacks? *IEEE Security & Privacy*, 1(3):60–62, 2003.

[108] Ian Foster. What is the Grid? A Three Point Checklist, July 2002.

[109] R.S. Francis and I.D. Mathieson. A benchmark parallel sort for shared memory multiprocessors. *IEEE Transactions on Computers*, 37(12):1619–1626, 1988.

[110] Cédric Févotte and Jérôme Idier. Algorithms for Nonnegative Matrix Factorization with the β-Divergence. *Neural Computation*, 23(9):2421–2456, 2011.

[111] Karthik Ganesan, Jungho Jo, W. Lloyd Bircher, Dimitris Kaseridis, Zhibin Yu, and Lizy K John. System-level Max POwer (SYMPO) - a systematic approach for escalating system-level power consumption using synthetic benchmarks. In *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 19–28, 2010.

[112] Karthik Ganesan and Lizy K. John. MAximum Multicore POwer (MAMPO): An Automatic Multithreaded Synthetic Power Virus Generation Framework for Multicore Systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 1–12, New York, NY, USA, 2011. Association for Computing Machinery.

[113] Craig Gentry. *A fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, 2009.

[114] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. Association for Computing Machinery.

[115] Craig Gentry. Computing Arbitrary Functions of Encrypted Data. *Commun. ACM*, 53(3):97–105, March 2010.

[116] Sanjay Ghemawat and Jeff Dean. LevelDB. `https://github.com/google/leveldb`, March 2011. Accessed on 28.01.2022.

[117] GlobalPlatform, Inc. *TEE Client API Specification Version 1.0*, July 2010. GPD_SPE_007.

[118] GlobalPlatform, Inc. *TEE Sockets API Specification*, January 2017. GPD_SPE_100.

[119] GlobalPlatform, Inc. *Root of Trust Definitions and Requirements Version 1.1*, June 2018. GP_REQ_025.

[120] GlobalPlatform, Inc. *TEE Internal Core API Specification Version 1.2*, October 2018. GPD_SPE_010.

[121] GlobalPlatform, Inc. *TEE System Architecture Version 1.2*, November 2018. GPD_SPE_009.

[122] Google LLC. Introducing Google Cloud Confidential Computing with Confidential VMs. `https://cloud.google.com/blog/products/identity-security/introducing-google-cloud-confidential-computing-with-confidential-vms`, July 2020. Accessed on 28.01.2022.

[123] Christian Göttel, Pascal Felber, and Valerio Schiavoni. Developing Secure Services for IoT with OP-TEE: A First Look at Performance and Usability. In José Pereira and Laura Ricci, editors, *Distributed Applications and International Publishing*, pages 170–178, Cham, 2019. Springer International Publishing.

[124] Christian Göttel, Pascal Felber, and Valerio Schiavoni. iperfTZ: Understanding Network Bottlenecks for TrustZone-Based Trusted Applications. In Mohsen Ghaffari, Mikhail Nesterenko, Sébastien Tixeuil, Sara Tucci, and Yukiko Yamauchi, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 178–193, Cham, 2019. Springer International Publishing.

[125] Christian Göttel, Lars Nielsen, Niloofar Yazdani, Pascal Felber, Daniel E. Lucani, and Valerio Schiavoni. Hermes: Enabling Energy-Efficient IoT Networks with Generalized Deduplication. In *Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems*, DEBS '20, pages 133–136, New York, NY, USA, 2020. Association for Computing Machinery.

[126] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. Cache Attacks on Intel SGX. In *Proceedings of the 10th European Workshop on Systems Security*, EuroSec'17, pages 1–6, New York, NY, USA, 2017. Association for Computing Machinery.

[127] Laurent Goudet and T. Demarne. 1g edition, August 2012. AL/LG/0838/003.

[128] Great Internet Mersenne Prime Search. Mersenne prime search software.

[129] Peter Greenhalgh. big.LITTLE Processing with ARM Cortex-A15 & Coretex-A7. White Paper, September 2011.

[130] John L. Gustafson. Reevaluating Amdahl's Law. *Commun. ACM*, 31(5):532–533, May 1988.

[131] Christian Göttel, Konstantinos Parasyris, Osman Unsal, Pascal Felber, Marcelo Pasin, and Valerio Schiavoni. Scrooge Attack: Undervolting ARM Processors for Profit: Practical experience report. In *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*, pages 187–197, 2021.

[132] S. Haber and W.S. Stornetta. How to time-stamp a digital document. *J. Cryptology*, 3(2):99–111, January 1991.

[133] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[134] Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Side Channels in Cloud Services: Deduplication in Cloud Storage. *IEEE Security & Privacy*, 8(6):40–47, 2010.

[135] Jarvis Haupt, Waheed U. Bajwa, Michael Rabbat, and Robert Nowak. Compressed Sensing for Networked Data. *IEEE Signal Processing Magazine*, 25(2):92–101, 2008.

[136] Mike Hearn. Corda: A distributed ledger. Technical report, R3, November 2016.

[137] John L. Hennessy and David A. Patterson. *Computer Architecture - A Quantitative Approach.* Morgan Kaufmann, fifth edition, 2012.

[138] Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming.* Morgan Kaufmann, first edition, 2008.

[139] Felicitas Hetzelt and Robert Buhren. Security Analysis of Encrypted Virtual Machines. In *Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '17, pages 129–142, New York, NY, USA, 2017. Association for Computing Machinery.

[140] Felicitas Hetzelt, Martin Radev, Robert Buhren, Mathias Morbitzer, and Jean-Pierre Seifert. VIA: Analyzing Device Interfaces of Protected Virtual Machines. In *Annual Computer Security Applications Conference*, ACSAC, pages 273–284, New York, NY, USA, 2021. Association for Computing Machinery.

[141] Hex Five Security. *MultiZone Security*, 1.6 edition, October 2021. Accessed on 23.11.2021.

[142] Mark D. Hill. What is Scalability? *SIGARCH Comput. Archit. News*, 18(4):18–21, December 1990.

[143] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[144] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. Using Innovative Instructions to Create Trustworthy Software Solutions. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, pages 1–8, New York, NY, USA, 2013. Association for Computing Machinery. Article No. 11.

[145] Owen S. Hofmann, Sangman Kim, Alan M. Dunn, Michael Z. Lee, and Emmett Witchel. InkTag: Secure Applications on an Untrusted Operating System. *SIGARCH Comput. Archit. News*, 41(1):265–278, March 2013.

[146] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 79, pages 2554–2558, 1982.

[147] Mark Horowitz, Thomas Indermaur, and Ricardo Gonzalez. Low-Power Digital Design. In *Proceedings of 1994 IEEE Symposium on Low Power Electronics*, pages 8–11, 1994.

[148] Kien A. Hua, Ning Jiang, Jason Kuhns, Vaithiyanathan Sundaram, and Cliff Zou. Redundancy control through traffic deduplication. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 10–18, 2015.

[149] Joseph Hurley and Dag Johansen. Self-Managing Data in the Clouds. In *2014 IEEE International Conference on Cloud Engineering*, pages 417–423, 2014.

[150] Intel Berkeley Research lab. Intel Lab Data. `http://db.csail.mit.edu/labdata/labdata.html`, April 2004. Accessed on 28.01.2022.

[151] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual*, 325384-075us edition, June 2021.

[152] International Business Machines Corporation. Compaq, Hewlett Packard, IBM, Intel, and Microsoft Announce Open Alliance to Build Trust and Security into PCs for e-business. `https://www-03.ibm.com/press/us/en/pressrelease/2016.wss`, October 1999. Accessed on 29.11.2014.

[153] International Business Machines Corporation. *Secure Service Container User's Guide*, sc28-6971-01 edition, 2017.

[154] International Business Machines Corporation. Data-in-use protection on IBM Cloud using Intel SGX. `https://www.ibm.com/cloud/blog/data-use-protection-ibm-cloud-using-intel-sgx`, May 2018. Accessed on 28.01.2022.

[155] International Business Machines Corporation. Leveraging AMD SEV in the IBM Hybrid Cloud. `https://www.ibm.com/blogs/research/2020/11/amd-sev-ibm-hybrid-cloud/`, 11 2020. Accessed on 28.01.2022.

[156] International Electrotechnical Commission. *IEC 60320-2-4:2018 Appliance couplers for household and similar general purposes*, 2.0 edition, June 2018.

[157] Pooyan Jamshidi, Claus Pahl, Nabor C. Mendonça, James Lewis, and Stefan Tilkov. Microservices: The Journey So Far and Challenges Ahead. *IEEE Software*, 35(3):24–35, 2018.

[158] Abdallah Jarwan, Ayman Sabbah, and Mohamed Ibnkahla. Data transmission reduction schemes in wsns for efficient iot systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1307–1324, 2019.

[159] JEDEC. *Embedded Multi-Media Card (eMMC), Electrical Standard (5.1)*, 5.1 edition, January 2019.

[160] Simon P. Johnson, Uday R. Savagaonkar, Vincent R. Scarlata, Francis X. McKeen, and Carlos V. Rozas. Technique for supporting multiple secure enclaves, December 2010. United States Patent 8,972,746.

[161] Elizabeth K. Joseph. Technical overview of Secure Execution for Linux on IBM Z. `https://developer.ibm.com/blogs/technical-overview-of-secure-execution-for-linux-on-ibm-z/`, April 2020. Accessed on 28.01.2022.

[162] Manolis Kaliorakis, Athanasios Chatzidimitriou, George Papadimitriou, and Dimitris Gizopoulos. Statistical Analysis of Multicore CPUs Operation in Scaled Voltage Conditions. *IEEE Computer Architecture Letters*, 17(2):109–112, 2018.

[163] David Kaplan. Protecting VM Register State with SEV-ES. `https://support.amd.com/TechDocs/Protecting%20VM%20Register%20State%20with%20SEV-ES.pdf`, February 2017. Accessed on 28.01.2022.

[164] David Kaplan, Jeremy Powell, and Tom Woller. AMD Memory Encryption. `https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf`, April 2016. Accessed on 28.01.2022.

[165] Alan C. Kay. A Personal Computer for Children of All Ages. In *Proceedings of the ACM Annual Conference - Volume 1*, ACM '72, New York, NY, USA, 1972. Association for Computing Machinery.

[166] Zijo Kenjar, Tommaso Frassetto, David Gens, Michael Franz, and Ahmad-Reza Sadeghi. V0LTpwn: Attacking x86 Processor Integrity from Software. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1445–1461. USENIX Association, August 2020.

[167] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, 11:5–38, January 1883.

[168] Sunny King and Scott Nadal. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake. `https://www.peercoin.net/whitepapers/peercoin-paper.pdf`, August 2012. Accessed on 28.01.2022.

[169] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[170] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[171] Jonathan Koomey, Stephen Berard, Marla Sanchez, and Henry Wong. Implications of Historical Trends in the Electrical Efficiency of Computing. *IEEE Annals of the History of Computing*, 33(3):46–54, 2011.

[172] Panos Koutsovasilis, Konstantinos Parasyris, Christos D. Antonopoulos, Nikolaos Bellas, and Spyros Lalis. Dynamic undervolting to improve energy efficiency on multicore x86 cpus. *IEEE Transactions on Parallel and Distributed Systems*, 31(12):2851–2864, 2020.

[173] Ulf Kulau, Felix Büsching, and Lars Wolf. Undervolting in wsns — a feasibility analysis. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 553–558, 2014.

[174] Kaz Kylheku. Kazlib. `http://www.kylheku.com/~kaz/kazlib.html`, November 2000. Accessed on 28.01.2022.

[175] Leslie Lamport. Proving the Correctness of Multiprocess Programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, 1977.

[176] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM*, 21(7):558–565, July 1978.

[177] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. Technical Report 54, Computer Science Laboratory, SRI International, March 1980.

[178] Ben Lapid and Avishai Wool. Cache-Attacks on the ARM TrustZone Implementations of AES-256 and AES-256-GCM via GPU-Based Analysis. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography – SAC 2018*, pages 235–256, Cham, 2019. Springer International Publishing.

[179] Etienne Le Sueur and Gernot Heiser. Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns. HotPower'10, pages 1–8, USA, 2010. USENIX Association.

[180] Kevin Lepak, Gerry Talbot, Sean White, Noah Beck, and Sam Naffziger. The Next Generation AMD Enterprise Server Product Architecture. `https://old.hotchips.org/wp-content/uploads/hc_archives/hc29/HC29.22-Tuesday-Pub/HC29.22.90-Server-Pub/HC29.22.921-EPYC-Lepak-AMD-v2.pdf`, August 2017. Accessed on 28.01.2022.

[181] Peter T. Lewis. Congressional back caucus foundation 15th annual legislative weekend, September 1985. Washington, D.C.

[182] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 717–732. USENIX Association, August 2021.

[183] Yanlin Li, Jonathan McCune, James Newsome, Adrian Perrig, Brandon Baker, and Will Drewry. MiniBox: A Two-Way Sandbox for x86 Native Code. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 409–420, Philadelphia, PA, June 2014. USENIX Association.

[184] J.C.R. Licklider and Robert W. Taylor. The Computer as a Communication Device. *Science and Technology*, April 1968.

[185] ARM Limited. Speculation Barrier. `https://github.com/ARM-software/speculation-barrier`, May 2018. Accessed on 28.01.2022.

[186] ARM Limited. Cache Speculation Side-channels. Whitepaper, June 2020. Version 2.5.

[187] Linar Limited. Open Portable Trusted Execution Environment. `https://www.op-tee.org/`. Accessed on 28.01.2022.

[188] Linaro Limited. Trusted Firmware. `https://www.trustedfirmware.org/`. Accessed on 28.01.2022.

[189] Linaro Limited. OP-TEE Sanity Testsuite. `https://github.com/OP-TEE/optee_test/tree/3.2.0`, June 2018. Accessed on 25.11.2021.

[190] Joshua Lind, Ittay Eyal, Florian Kelbert, Oded Naor, Peter R. Pietzuch, and Emin Gün Sirer. Teechain: Scalable Blockchain Payments using Trusted Execution Environments. *CoRR*, abs/1707.05454, 2017.

[191] LINDY. *iPower Control 2x6M/2x6XM User Manual*, first edition, June 2015. No. 32660 / 32690.

[192] Moritz Lipp, Daniel Gruss, and Michael Schwarz. AMD Prefetch Attacks through Power and Time. In *31th USENIX Security Symposium (USENIX Security 22)*. USENIX Association, August 2022.

[193] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[194] Tencent Holdings Ltd. 例格. `https://cloud.tencent.com/document/product/213/11518#M6ce`, November 2021. Accessed on 28.01.2022.

[195] Y. Léger. Public Preview. `https://blog.scaleway.com/online-labs-public-preview`, October 2014. Accessed on 28.01.2022.

[196] Berenice Mann. Architecting a more Secure world with isolation and virtualization. `https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/architecting-more-secure-world-with-isolation-and-virtualization`, August 2018. Accessed on 28.01.2022.

[197] Francesco Marcelloni and Massimo Vecchio. A Simple Algorithm for Data Compression in Wireless Sensor Networks. *IEEE Communications Letters*, 12(6):411–413, 2008.

[198] Patrick Marlier and Mascha Kurpicz. powerspy.py. `https://github.com/patrickmarlier/powerspy.py`, February 2016. Accessed on 28.01.2022.

[199] Emmanouil Maroudas, Spyros Lalis, Nikolaos Bellas, and Christos D. Antonopoulos. Exploring the Potential of Context-Aware Dynamic CPU Undervolting. In *Proceedings of the 18th ACM International Conference on Computing Frontiers*, CF '21, pages 73–82, New York, NY, USA, 2021. Association for Computing Machinery.

[200] Hector Martin. Twitter Thread. `https://twitter.com/marcan42/status/1080869868889501696`, January 2019. Accessed on 28.01.2022.

[201] Avantika Mathur, Mingming Cao, and Suparna Bhattacharya. The new ext4 filesystem: current status and future plans. *Proceedings of the Linux Symposium*, 2:21–33, June 2007.

[202] Friedemann Mattern and Christian Floerkemeier. *From the Internet of Computers to the Internet of Things*, pages 242–259. Springer Berline Heidelberg, Berlin, Heidelberg, 2010.

[203] MaxLinear. *MaxLinear Data Sheet MxL7704 Five Output Universal PMIC*, 1b edition, July 2018.

[204] Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil Gligor, and Adrian Perrig. TrustVisor: Efficient TCB Reduction and Attestation. In *2010 IEEE Symposium on Security and Privacy*, pages 143–158, 2010.

[205] Jonathan M. McCune, Bryan J. Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki. Flicker: An Execution Infrastructure for TCB Minimization. volume 42, pages 315–328, New York, NY, USA, April 2008. Association for Computing Machinery.

[206] Francis X. McKeen, Carlos V. Rozas, Uday R. Savagaonkar, Simon P. Johnson, Vincent Scarlata, Michael A. Goldsmith, Ernie Brickell, Jiang Tao Li, Howard C. Herbert, Prashant Dewan, Stephen J. Tolopka, Gilbert Neiger, David Durham, Gary Graunke, Bernard Lint, Don A. Van Dyke, Joseph Cihula, Stalinselvaraj Jeyasingh, Stephen R. Van Doren, Dion Rodgers, John Garney, and Asher Altman. Method and apparatus to provide secure application execution. United States Patent 9,087,200, June 2012.

[207] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative Instructions and Software Model for Isolated Execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, pages 1–8, New York, NY, USA, 2013. Association for Computing Machinery. Article No. 10.

[208] Robert J. McMahon. iperf2. `https://sourceforge.net/projects/iperf2/`. Accessed on 28.01.2022.

[209] A. Melnikov and K. Zeilenga. Simple Authentication and Security Layer (SASL). RFC4422, June 2006.

[210] Galit Mendelson. All You Need To Know About Power over Ethernet (PoE) and the IEEE 802.3af Standard. Technical report, PowerDsine, June 2004. White Paper.

[211] Ralph C. Merkle. *Secrecy, authenticaation, and public key systems*. PhD thesis, Electrical Engineering, Stanford, June 1979. Technical Report No. 1979-1.

[212] Ralph C. Merkle. United States Patent 4,309,569, September 1979. Issued 05.01.1982.

[213] Ralph C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In Carl Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, pages 369–378, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.

[214] Dutch T. Meyer and William J. Bolosky. A Study of Practical Deduplication. In *9th USENIX Conference on File and Storage Technologies (FAST 11)*, San Jose, CA, February 2011. USENIX Association.

[215] Microsoft. The Coco Framework. `https://github.com/Azure/coco-framework/blob/7de1519835ed7e851f6ffe17cca8e5690ce62e0d/docs/CocoFrameworkwhitepaper.pdf`, August 2017. Accessed on 28.01.2022.

[216] Microsoft Corporation. Azure and AMD connounce landmark in confidential computing evolution. `https://azure.microsoft.com/en-us/blog/azure-and-amd-enable-lift-and-shift-confidential-computing/`, March 2021. Accessed on 28.01.2022.

[217] Microsoft Corporation. DCsv2-series. `https://docs.microsoft.com/en-us/azure/virtual-machines/dcv2-series`, February 2021. Accessed on 28.01.2022.

[218] Microsoft Corporation. DCsv3 and DCdsv3-series. `https://docs.microsoft.com/en-us/azure/virtual-machines/dcv3-series`, April 2021. Accessed on 28.01.2022.

[219] S. Millman, editor. *A History of Engineering and Science in the Bell System, Physical Sciences (1925-1980)*. AT&T Bell Laboratories, 1983.

[220] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–118, April 1965.

[221] Mathias Morbitzer. Scanclave: Verifying Application Runtime Integrity in Untrusted Environments. In *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 198–203, 2019.

[222] Mathias Morbitzer, Manuel Huber, and Julian Horsch. Extracting Secrets from Encrypted Virtual Machines. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, CODASPY '19, pages 221–230, New York, NY, USA, 2019. Association for Computing Machinery.

[223] Mathias Morbitzer, Manuel Huber, Julian Horsch, and Sascha Wessel. SEVered: Subverting AMD's Virtual Machine Encryption. In *Proceedings of the 11th European Workshop on Systems Security*, EuroSec'18, New York, NY, USA, 2018. Association for Computing Machinery.

[224] Mathias Morbitzer, Sergej Proskurin, Martin Radev, Marko Dorfhuber, and Erick Quintanar Salas. SEVerity: Code Injection Attacks against Encrypted Virtual Machines. In *arXiv*, pages 1–12, 2021.

[225] Christina Müller. Execution of Smart Contracts with ARM TrustZone. Master's thesis, University of Bern, September 2019.

[226] Christina Müller. Hyperledger Fabric chaincode execution with OP-TEE. `https://github.com/piachristel/open-source-fabric-optee-chaincode`, September 2019. Accessed on 28.01.2022.

[227] Christina Müller, Marcus Brandenburger, Christian Cachin, Pascal Felber, Christian Göttel, and Valerio Schiavoni. TZ4Fabric: Executing Smart Contracts with ARM TrustZone : (Practical Experience Report). In *2020 International Symposium on Reliable Distributed Systems (SRDS)*, pages 31–40, 2020.

[228] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based Fault Injection Attacks against Intel SGX. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1466–1482, 2020.

[229] Mohamed Nabeel, Ning Shang, and Elisa Bertino. Privacy-Preserving Filtering and Covering in Content-Based Publish Subscribe Systems. Technical Report 15, Purdue University, West Lafayette, Indiana, USA, 2009.

[230] Mohamed Nabeel, Ning Shang, and Elisa Bertino. Efficient Privacy Preserving Content Based Publish Subscribe Systems. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, SACMAT '12, pages 133–144, New York, NY, USA, 2012. Association for Computing Machinery.

[231] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. `https://bitcoin.org/bitcoin.pdf`, October 2008. Accessed on 28.01.2022.

[232] Roger M. Needham and Michael D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Commun. ACM*, 21(12):993–999, December 1978.

[233] Lars Nielsen, Dorian Burihabwa, Valerio Schiavoni, Pascal Felber, and Daniel E. Lucani. Minervafs: A user-space file system for generalised deduplication: (practical experience report). In *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*, pages 254–264, 2021.

[234] Lars Nielsen, Rasmus Vestergaard, Niloofar Yazdani, Prasad Talasila, Daniel E. Lucani, and Marton Sipos. Alexandria: A Proof-of-Concept Implementation and Evaluation of Generalised Data Deduplication. In *2019 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6, 2019.

[235] NLANR/DAST. iperf. `https://sourceforge.net/projects/iperf/`. Accessed on 28.01.2022.

[236] Adel Noureddine. Joular PowerSpyCli. `https://github.com/joular/powerspycli`, September 2021. Accessed on 28.01.2022.

[237] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The Akamai Network: A Platform for High-Performance Internet Applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19, August 2010.

[238] OASIS Open. *OASIS Advanced Message Queuing Protocol (AMQP)*, 1.0 edition, October 2012.

[239] OASIS Open. *MQTT*, 5.0 edition, March 2019.

[240] OMTP. *Advanced Trusted Environment: OMTP TR1*. Open Mobile Terminal Platform Ltd., v1.1 edition, May 2009.

[241] OMTP. *Trusted Environment: OMTP TR0*. Open Mobile Terminal Platform Ltd., v1.2 edition, May 2009.

[242] Emmanuel Owusu, Jorge Guajardo, Jonathan McCune, Jim Newsome, Adrian Perrig, and Amit Vasudevan. OASIS: On Achieving a Sanctuary for Integrity and Secrecy on Untrusted Platforms. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 13–24, New York, NY, USA, 2013. Association for Computing Machinery.

[243] Christof Paar and Jan Pelzl. *Understanding Cryptography*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2nd edition, 2010.

[244] George Papadimitriou, Athanasios Chatzidimitriou, and Dimitris Gizopoulos. Adaptive Voltage/Frequency Scaling and Core Allocation for Balanced Energy and Performance on Multicore CPUs. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 133–146, 2019.

[245] George Papadimitriou, Athanasios Chatzidimitriou, Manolis Kaliorakis, Yannos Vastakis, and Dimitris Gizopoulos. Micro-Viruses for Fast System-Level Voltage Margins Characterization in Multicore CPUs. In *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 54–63, 2018.

[246] George Papadimitriou, Manolis Kaliorakis, Athanasios Chatzidimitriou, Dimitris Gizopoulos, Peter Lawthers, and Shidhartha Das. Harnessing Voltage Margins for Energy Efficiency in Multicore CPUs. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, pages 503–516, New York, NY, USA, 2017. Association for Computing Machinery.

[247] George Papadimitriou, Manolis Kaliorakis, Athanasios Chatzidimitriou, Charalampos Magdalinos, and Dimitris Gizopoulos. Voltage margins identification on commercial x86-64 multicore microprocessors. In *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 51–56, 2017.

[248] Bryan Parno, Jacob R. Lorch, John R. Douceur, James Mickens, and Jonathan M. McCune. Memoir: Practical State Continuity for Protected Modules. In *2011 IEEE Symposium on Security and Privacy*, pages 379–394, 2011.

[249] Pankesh Patel, Ajith H. Ranabahu, and Amit P. Sheth. Service Level Agreement in Cloud Computing. `https://corescholar.libraries.wright.edu/knoesis/78/`, 2009. Accessed on 28.01.2022.

[250] Robert Pettersen, Håvard D. Johansen, and Dag Johansen. Secure Edge Computing with ARM TrustZone. In *Proceedings of the 2nd International COnference on Internet of Things, Big Data and Security - IoTBDS*, pages 102–109. INSTICC, SciTePress, 2017.

[251] Zahra Pooranian, Kang-Cheng Chen, Chia-Mu Yu, and Mauro Conti. RARE: Defeating side channels based on data-deduplication in cloud storage. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 444–449, 2018.

[252] Serguei Popov. The tangle. Whitepaper, April 2016.

[253] Nelly Porter, Jason Garms, and Sergey Simakov. Introducing Asylo: an open-source framework for confidential computing. `https://cloud.google.com/blog/products/identity-security/introducing-asylo-an-open-source-framework-for-confidential-computing`, May 2018. Accessed on 28.01.2022.

[254] Louis Pouzin. Presentation and Major Design Aspects of the Cyclades Computer Network. In *Proceedings of the NATO Advanced Study Institute on Computer Communication Networks*, pages 415–434, Sussex, United Kingdom, 1973. Noordhoff International Publishing.

[255] Louis Pouzin. A Proposal fo Interconnecting Packet Switching Networks. In *EUROCOMP 74 Proceedings*, Uxbridge, Middlesex, England, 1974. Brunel University.

[256] S.S. Pradhan and K. Ramchandran. Distributed source coding using syndromes (DISCUS): design and construction. *IEEE Transactions on Information Theory*, 49(3):626–643, 2003.

[257] Bernd Prünster, Gerald Palfinger, and Christian Paul Kollmann. Fides – Unleashing the Full Potential of Remote Attestation. In *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications*, volume 2: SECRYPT, pages 314–321. SciTePress - Science and Technology Publications, July 2019.

[258] QEMU Wiki. Documentation/Networking. `https://wiki.qemu.org/Documentation/Networking#User_Networking_.28SLIRP.29`. Accessed on 28.01.2022.

[259] Pengfei Qiu, Dongsheng Wang, Yongqiang Lyu, and Gang Qu. VoltJockey: Breaching TrustZone by Software-Controlled Voltage Manipulation over Multi-Core Frequencies. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, pages 195–209, New York, NY, USA, 2019. Association for Computing Machinery.

[260] Michael O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[261] Martin Radev and Mathias Morbitzer. *Exploiting Interfaces of Secure Encrypted Virtual Machines*, pages 1–12. Association for Computing Machinery, New York, NY, USA, 2020.

[262] Raspberry Pi Foundation. PoE HAT. https://www.raspberrypi.com/products/poe-hat/, 2017. Accessed on 28.01.2022.

[263] I. S. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[264] David Reinsel, John Gantz, and John Rydning. The Digitization of the World From Edge to Core. https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf, November 2018. Accessed on 28.01.2022.

[265] F.H. Reynolds. Thermally accelerated aging of semiconductor components. *Proceedings of the IEEE*, 62(2):212–222, 1974.

[266] Lawrence G. Roberts. Multiple Computer Networks and Intercomputer Communication. In *Proceedings of the First ACM Symposium on Operating System Principles*, SOSP'67, pages 3.1–3.6, New York, NY, USA, 1967. Association for Computing Machinery.

[267] Dan Rosenberg. QSEE TrustZone Kernel Integer Overflow Vulnerability. black hat 2014, July 2014.

[268] Mark Russinovich, Edward Ashton, Christine Avanessians, Miguel Castro, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, Cédric Fournet, Matthew Kerner, Sid Krishna, Julien Maffre, Thomas Moscibroda, Kartik Nayak, Olga Ohrimenko, Felix Schuster, Roy Schuster, Alex Shamis, Olga Vrousgou, and Christoph M. Wintersteiger. CCF: A Framework for Building Confidential Verifiable Replicated Services. Technical report, Microsoft Research & Microsoft Azure, April 2019.

[269] Keegan Ryan. Hardware-Backed Heist: Extracting ECDSA Keys from Qualcomm's TrustZone. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, pages 181–194, New York, NY, USA, 2019. Association for Computing Machinery.

[270] Reiner Sailer, Zhang Xiaolan, Trent Jaeger, and Leendert van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *Proceedings of the 13th USENIX Security Symposium*, pages 1–16, USA, August 2004. USENIX Association.

[271] Shruti Sanadhya, Raghupathy Sivakumar, Kyu-Han Kim, Paul Congdon, Sriram Lakshmanan, and Jatinder Pal Singh. Asymmetric Caching: Improved Network Deduplication for Mobile Devices. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, pages 161–172, New York, NY, USA, 2012. Association for Computing Machinery.

[272] R.A. Scantlebury and P.T. Wilkinson. The National Physical Laboratory Data Communications Network. In *Proceedings of the 2nd ICCC 74*, pages 223–228, 1974.

[273] Fred B. Schneider. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Comput. Surv.*, 22(4):299–319, December 1990.

[274] Bruce Schneier, Doug Whiting, David Wagner, and Chris Hall. Twofish: A 128-Bit Block Cipher, June 1998.

[275] C.L. Schuba, I.V. Krsul, M.G. Kuhn, E.H. Spafford, A. Sundaram, and D. Zamboni. Analysis of a denial of service attack on TCP. In *Proceedings. 1997 IEEE Symposium on Security and Privacy*, pages 208–223, 1997.

[276] Srinath Setty, Soumya Basu, Lidong Zhou, Justine Stephenson, and Ramarathnam Venkatesan. Enabling secure and resource-efficient blockchain networks with VOLT. Technical Report MSR-TR-2017-38, Microsoft, August 2017. `https://www.microsoft.com/en-us/research/publication/enabling-secure-resource-efficient-blockchain-networks-volt/`.

[277] Y. Shafranovich. Common Formand and MIME Type for Comma-Separated Values (CSV) Files. RFC4180, October 2005.

[278] Claude E. Shannon. A MAthematical Theory of Cryptography. Technical report, Bell Telephone Laboratories, September 1945.

[279] Claude E. Shannon. Communication theory of secrecy systems. *The Bell System Technical Journal*, 28(4):656–715, 1949.

[280] Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). RFC7252, June 2014.

[281] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[282] Terry Slattery. The origin of ttcp (test TCP). `https://netcraftsmen.com/the-origin-of-ttcp-test-tcp/`, August 2007. Accessed on 28.01.2022.

[283] Neil T. Spring and David Wetherall. A Protocol-Independent Technique for Eliminating Redundant Network Traffic. *SIGCOMM Comput. Commun. Rev.*, 30(4):87–95, August 2000.

[284] Lara Srivastava, Phillippa Biggs, Tim Kelly, Youlia Lozanova, Lilia Pérez Chavolla, Jaroslaw Ponder, Raushan Sagalbayeva, Svetlana Skvortsova, and Christine Sund. The Internet of Things. Technical report, International Telecommunication Union, Geneva, Switzerland, November 2005.

[285] Julian James Stephen, Savvas Savvides, Vinaitheerthan Sundaram, Masoud Saeida Ardekani, and Patrick Eugster. STYX: Stream Processing with Trustworthy Cloud-Based Execution. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, pages 348–360. Association for Computing Machinery, 2016.

[286] Nick Szabo. Smart Contracts. https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html, 1994. Accessed on 28.01.2022.

[287] Andrew S. Tanenbaum and Albert S. Woodhull. *Operating Systems - Design and Implementation*. Pearson Education, third edition, 2009.

[288] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1057–1074, Vancouver, BC, August 2017. USENIX Association.

[289] The Santa Cruz Operation, Inc. *System V Application Binary Interface*, 4.1 edition, 1997.

[290] Trusted Computing Group, Inc. *Trusted Platform Module Library - Part 1: Architecture*, 00.96 edition, March 2013.

[291] Trusted Computing Platform Alliance (TCPA). *Main Specification*, version 1.1b edition, February 2002.

[292] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(5):345–363, 1936.

[293] Ubiquiti Networks. *Managed PoE+ Gigabit Switches with SFP*, 2018. AJ_JL041618.

[294] Eben Upton. Raspberry Pi 3 Model B+ on sale now at $35. https://www.raspberrypi.com/news/raspberry-pi-3-model-bplus-sale-now-35/, March 2018. Accessed on 28.01.2022.

[295] L. Upton. Raspberry Pi colocation. https://www.raspberrypi.org/blog/raspberry-pi-colocation, April 2013. Accessed on 28.01.2022.

[296] Amit Vasudevan, Emmanuel Owusu, Zongwei Zhou, James Newsome, and Jonathan M. McCune. Trustworthy Execution on Mobile Devices: What Security Properties Can My Mobile Platform Give Me? In Stefan Katzenbeisser, Edgar Weippl, L. Jean Camp, Melanie Volkamer, Mike Reiter, and Xinwen Zhang, editors, *Trust and Trustworthy Computing*, pages 159–178, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[297] Sébastien Vaucher, Niloofar Yazdani, Pascal Felber, Daniel E. Lucani, and Valerio Schiavoni. ZipLine: In-Network Compression at Line Speed. In *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '20, pages 399–405, New York, NY, USA, 2020. Association for Computing Machinery.

[298] Rasmus Vestergaard, Daniel E. Lucani, and Qi Zhang. Generalized Deduplication: Lossless Compression for Large Amounts of Small IoT Data. In *European Wireless 2019; 25th European Wireless Conference*, pages 1–5, 2019.

[299] Steve Vinoski. Advanced Message Queuing Protocol. *IEEE Internet Computing*, 10(6):87–89, 2006.

[300] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 Protocol. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 1–12, Oakland, CA, November 1996. USENIX Association.

[301] Govind Wathan. Arm DynamIQ: Technology for the next era of compute. `https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/arm-dynamiq-technology-for-the-next-era-of-compute`, March 2017. Accessed on 28.01.2022.

[302] Nico Weichbrodt, Anil Kurmus, Peter Pietzuch, and Rüdiger Kapitza. AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves. In Ioannis Askoxylakis, Sotiris Ioannidis, Sokratis Katsikas, and Catherine Meadows, editors, *Computer Security – ESORICS 2016*, pages 440–457, Cham, 2016. Springer International Publishing.

[303] Christof Weinhardt, Arun Anandasivam, Benjamin Blau, Nikolay Borissov, Thomas Meinl, Wibke Michalk, and Jochen Stößer. Cloud Computing – A Classification, Business Models, and Research Directions. *Business & Information Systems Engineering*, 1(5):391–399, September 2009.

[304] Mark Weiser. The Computer of the 21st Century. *Scientific American Special Issue: Communications, Computers, and Networks*, 265(3), September 1991.

[305] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for Reduced CPU Energy. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, OSDI '94, pages 1–11, USA, 1994. USENIX Association.

[306] T. Welch. A Technique for High-Performance Data Compression. *Computer*, 17(06):8–19, June 1984.

[307] Luca Wilke, Jan Wichelmann, Mathias Morbitzer, and Thomas Eisenbarth. SEVurity: No Security Without Integrity : Breaking Integrity-Free Memory Encryption with Minimal Assumptions, 2020.

[308] Eugene Wong. Retrieving dispersed data from SDD-1: A system for distributed databases. In *Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks*, pages 217–235, May 1977.

[309] A.D. Wood and J.A. Stankovic. Denial of Service in Sensor Networks. *Computer*, 35(10):54–62, 2002.

[310] Mou Wu, Liansheng Tan, and Naixue Xiong. Data prediction, compression, and recovery in clustered wireless sensor networks for environmental monitoring applications. *Information Sciences*, 329:800–818, 2016. Special issue on Discovery Science.

[311] Yi Yang, Xinran Wang, Sencun Zhu, and Guohong Cao. SDAP: A Secure Hop-by-Hop Data Aggregation Protocol for Sensor Networks. In *Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '06, pages 356–367, New York, NY, USA, 2006. Association for Computing Machinery.

[312] Niloofar Yazdani and Daniel E. Lucani. Revolving Codes: High Performance and Low Overhead Network Coding. In *2019 IEEE 2nd Wireless Africa Conference (WAC)*, pages 1–5, 2019.

[313] Ning Zhang, Kun Sun, Deborah Shands, Wenjing Lou, and Y. Thomas Hou. TruSense: Information Leakage from TrustZone. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 1097–1105, 2018.

[314] Yupu Zhang, Abhishek Rajimwale, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. End-to-End Data Integrity for File Systems: A ZFS Case Study. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, FAST'10, pages 29–42, USA, 2010. USENIX Association.

[315] Bing Zhou and Jiangtao Wen. Efficient File Communication via Deduplication over Networks with Manifest Feedback. *IEEE Communications Letters*, 18(1):94–97, 2014.

[316] Yazhou Zu, Charles R. Lefurgy, Jingwen Leng, Matthew Halpern, Michael S. Floyd, and Vijay Janapa Reddi. Adaptive guardband scheduling to improve system-level efficiency of the POWER7+. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 308–321, 2015.