DIPLOMA THESIS

Proof-Of-Concept For A Smart Grid Controller Blockchain Platform For Virtual Power Plants

With Focus On Blockchain-Integrated Privacy Technologies

Submitted at the Faculty of Electrical Engineering and Information Technology, TU Wien
in partial fulfillment of the requirements for the degree of
Diplom-Ingenieur (equals Master of Sciences)

under supervision of

Ao. Univ.Prof. Dipl.-Ing. Dr.techn. Thilo Sauter
Univ.Ass. Dipl.-Ing. Marcus Meisel, Bakk.techn.

by

Andrija Goranović
Matr.Nr. 1225182
Wohnpark 2/4, 7022 Schattendorf

Vienna, January 21st 2019 _____

Kurzfassung

Steigender Energieverbrauch und der Klimawandel stellen globale Herausforderungen dar. Die Europäische Union hat in ihrer Klima- und Energiepolitik Ziele festgelegt, die sie bis 2030 zu erfüllen versucht. Neben der Verringerung der Treibhausgasemissionen um mindestens 40 % im Vergleich zu 1990 sollte der Anteil erneuerbarer Energiequellen auf mindestens 27 % erhöht werden und die Energieeffizienz um mindestens 27 %.

Die Umstellung von fossilen Brennstoffen auf erneuerbare Energiequellen schafft neue Chancen, aber auch Herausforderungen. Für die Integration erneuerbarer Energiequellen wurden verschiedene Konzepte entwickelt, wie beispielsweise Microgrids oder virtuelle Kraftwerke, die ständig untersucht und verbessert werden. Wesentlich für diese Ansätze ist jedoch der Ausbau des bestehenden Energienetzes mittels Kommunikationstechnik zu einem intelligenten Stromnetz, dem Smart Grid. Der Endbenutzer übernimmt eine aktive Rolle in einem Smart Grid, z. B.: können Lastspitzen können in Zusammenarbeit reduziert werden. Dies wird durch die Digitalisierung der EndbenutzerInnen mit intelligenten Stromzählern, den Smart Metern, und dem Übergang zum Smart Home erleichtert.

Eine vielversprechende Technologie, die derzeit in Smart Grids erforscht wird, ist die Blockchain, mit der verschiedene Prozesse in Smart Grids realisiert werden, die sich durch eine dezentrale Struktur auszeichnen. Die bekannteste Anwendung ist der lokale P2P-Energiehandel.

Das Ziel der Diplomarbeit war, eine Blockchain-Plattform als Proof-of-Concept aufzubauen, deren Schwerpunkt auf der Protokollierung der Energieerzeugung, des Energieverbrauchs und des Energiehandels liegt. Die TeilnehmerInnen der Plattform können ein Portfolio verschiedener Anlagen besitzen, beispielsweise aus Photovoltaikanlagen, Batterien, Elektroautos und Dieselgeneratoren.

In einer realen Umgebung wären die TeilnehmerInnen mit intelligenten Stromzählern oder IoT-Geräten ausgestattet, die mit der Plattform verbunden sind. Eine zentrale Instanz, die von einem Energieunternehmen oder einer Gemeinde kontrolliert wird, würde für die Verwaltung der Plattform verantwortlich sein, d.h., verantwortlich für die Registrierung der Smart Meter und IoT-Geräte sowie für die Weiterverarbeitung der Daten. Die Blockchain-Plattform ist erweiterbar, so dass auf deren Basis andere Anwendungen realisiert werden können, z. B.: die Begleichung von Pönalezahlungen bei Nichteinhaltung der Spannungsgrenzen. Aufgrund des Inkrafttretens der Datenschutz-Grundverordnung (DSGVO) am 25. Mai 2018 in der Europäischen Union liegt ein besonderer Schwerpunkt dieser Diplomarbeit auf dem Datenschutz.

Für die Validierung der Funktionen des Proof-of-Concept, wurde eine Modellregion mit Raspberry Pis simuliert. Um ein realistisches Szenario zu simulieren, wurden Last- und Erzeugungsprofile verwendet. Mit Hilfe der realisierten Plattform ließ sich eine Bewertung in Bezug auf Performance und Datenschutz durchführen. Abschließend werden Schlussfolgerungen gezogen, um Empfehlungen für die Integration der Blockchain-Plattform in eine "reale" Umgebung geben zu können, z. B.: Großschönau, Green Energy Lab oder Act4.energy im Burgenland.

Abstract

Rising energy consumption and climate change pose global challenges. The European Union has set three targets in its climate and energy policy that it aims to meet by 2030. In addition to reducing greenhouse gas emissions by at least 40 % compared to 1990 levels, the share of renewable energy sources should be increased to at least 27 % and energy efficiency by at least 27 %.

The shift from fossil fuels to renewable sources of energy creates new opportunities, but also challenges. For the integration of renewable energy sources, different concepts have been developed, such as microgrids or virtual power plants, which are constantly being investigated and improved. Essential for these approaches, however, is the expansion of the existing energy network by means of communication technology to form an intelligent power grid, the Smart Grid. The end user assumes an active role in a smart grid, f.e., load peaks can be reduced in cooperation. This will be facilitated by the digitization of end users using smart meters and the transition to the smart home.

One promising technology, that is currently being explored in smart grids is the blockchain, which is used to realize different processes in smart grids, that are characterized by a decentralized structure, of which the best-known application is the local P2P energy trading.

The aim of the diploma thesis was to build a blockchain platform as a proof-of-concept whose focus is on the protocolling of energy production, energy consumption and energy trading. The participants of the platform can own a portfolio of different installations, such as photovoltaic systems, batteries, electric cars and diesel generators.

In a real-life environment, participants would be equipped with smart meters or IoT devices that interface with the platform. A central authority, under the control of an energy company or a municipality, would be responsible for the administration of the platform, i.a. responsible for the registration of the smart meters and IoT devices as well as the further processing of the data. The blockchain platform is extensible so that other applications can be realized, f.e., the settlement of penalty payments in case of non-compliance with voltage limits. Due to the enforcement of the General Data Protection Regulation (GDPR) on 25th May 2018 in the European Union, this thesis lays a special focus on privacy.

To validate the features of the proof-of-concept, a model region was simulated using Raspberry Pis. In order to simulate a realistic scenario, existing load and generation profiles were used. With the help of the implemented platform, an evaluation was made regarding performance and data protection. Finally, conclusions are drawn to be able to provide recommendations for integrating the blockchain platform into a real-life environment, such as Großschönau, Green Energy Lab, or Act4.energy in Burgenland.

Acknowledgements

Thank you!

Table of Contents

Abbreviations

ABAC	Attribute-Based Access Control
ACL	Access Control List
AES	Advanced Encryption Standard
AMCL	Apache Milagro Cryptographic Library
AMI	Advanced Metering Infrastructure
AMR	Automated Meter Reading
APCS	APCS Power Clearing and Settlement AG
API	Application Programming Interface
APT	Advanced Packaging Tool
ARMv6	ARM version 6
ARMv7	ARM version 7
BCCSP	Blockchain Crypto Service Provider
BFT	Byzantine Fault Tolerance
CA	Certificate Authority
CBC	Cipher Block Chaining
CC	Chaincode
cgroups	CPUset Control Groups
CLI	Command Line Interface
configtx	Configuration Transaction
CPU	Central Processing Unit
CRL	Certificate Revocation List
CSCC	Configuration System Chaincode
CVPP	Commercial Virtual Power Plant
DER	Decentralized Energy Resources
DES	Data Encryption Standard
DNS	Domain Name System
DSGVO	Datenschutz-Grundverordnung
DSM	Demand-Side Management
DSO	Distribution System Operator
E-Control	Energie-Control Austria für die Regulierung der Elektrizitäts- und Erdgaswirtschaft
ECDSA	Elliptic Curve Digital Signature Algorithm
EMS	Energy Management System
EPRI	Electric Power Research Institute
ESCC	Endorsement System Chaincode
EU	European Union
EV	Electric Vehicle
EVM	Ethereum Virtual Machine
EWF	Energy Web Foundation
GAVE	Municipality Großschönau As Virtual Energy Storage
Geth	Go-Ethereum

GDPR	General Data Protection Regulation
GND	Ground
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HDMI	High Definition Multimedia Interface
HLF	Hyperledger Fabric
HMAC	Keyed-Hash Message Authentication Code
HSM	Hardware Security Modules
HTTP	Hypertext Transfer Protocol
I2C	Inter-Integrated Circuit
IBM	International Business Machines Corporation
ICA	Intermediate Certificate Authority
ICS	IBM Container Services
ICT	Information And Communication Technology
IoT	Internet-Of-Things
IT	Internet Protocol
IT	Information Technology
IV	Initialization Vector
JSON	Java Script Object Notation
JVM	Java Virtual Machine
LAN	Local Area Network
LLL	Low-level Lisp-like Language
LSCC	Lifecycle System Chaincode
MSP	Membership Service Provider
NFS	Network File System
NIST	National Institute Of Standards And Technology
NPM	Natural Polyglot Machine
OLED	Organic Light Emitting Diode
OS	Operating System
OTG	On The Go
OU	Organizational Unit
QSCC	Query System Chaincode
P2P	Peer-to-Peer
PBFT	Practical Byzantine Fault Tolerance
PEM	Privacy Enhanced Mail
PET	Privacy Enhancing Technologies
PoET	Proof-Of-Elapsed-Time
PoS	Proof-Of-Stake
PoW	Proof-Of-Work
PV	Photovoltaic
PWM	Pulse-Width Modulation
RAM	Random Access Memory
RCA	Root Certificate Authority
RES	Renewable Energy Sources
RPI	Raspberry Pi
RPI2B	Raspberry Pi 2 Model B
RPI3B	Raspberry Pi 3 Model B
RSA	Rivest-Shamir-Adleman
SBFT	Simplified Byzantine Fault Tolerance
SD	Secure Digital
SCL	Serial Clock
SDA	Serial Data
SDK	Software Development Kit
SGC	Smart Grid Control

SGCBP	Smart Grid Controller Blockchain Platform
SGCCS	Smart Grid Controller Container Services
SHA	Secure Hash Algorithm
SignedCDS	Signed Chaincode Package
SSH	Secure Shell
SSID	Service Set Identifier
TAG-e	TransActive Grid Element
TLS	Transport Layer Security
TPSGA	Technology Platform Smart Grids Austria
TSO	Transmission System Operator
TVPP	Technical Virtual Power Plant
USB	Universal Serial Bus
VCC	Voltage At The Common Collector
VDEW	Verband der Elektrizitätswirtschaft e. V.
VM	Virtual Machine
VPP	Virtual Power Plant
VSCC	Validation System Chaincode
Wi-Fi	Wireless Fidelity (IEEE 802.11)
YAML	YAML Ain't Markup Language
zk-SNARKs	Zero-Knowledge Succinct Non-Interactive Arguments Of Knowledge
ZKP	Zero Knowledge Proof
ZSL	Zero-Knowledge Security Layer

1 Introduction

This chapter first introduces the motivation for this master thesis, beginning with the global energy challenges and required technologies to solve these. Second, the Smart Grid Control (SGC) project proposal and the deduced problem statement for the thesis are described. This work is concluded with the presentation of the approach of this thesis to solve this problem statement and the required steps.

1.1 Motivation

Rising energy consumption and climate change pose global challenges. The European Union (EU) has set three targets in its climate and energy policy [Cou14] that it should meet by 2030. In addition to reducing greenhouse gas emissions by at least 40 % compared to 1990 levels, the share of renewable energy sources (RES) should be increased to at least 27 % and energy efficiency increased by at least 27 %.

The shift from fossil fuels to renewable sources of energy creates new opportunities but also challenges. For the integration of renewable energy sources, different concepts have been developed, such as virtual power plants (VPP) (see Section 2.2.2), which are constantly being investigated and improved. Essential for these approaches, however, is the expansion of the existing energy network by means of communication technology to form an intelligent power grid, the smart grid. The end user assumes an active role in a smart grid, e.g., load peaks could be reduced in cooperation with the end user. This will be facilitated by the digitization of devices installed at end users using smart meters and the synergetic combination of the transition to the smart home done by end users on their own.

One promising technology currently being explored in smart grids is the Blockchain. In 2008, the Bitcoin white paper [Nak08] was released with the aim of creating a peer-to-peer (P2P) currency that would eliminate the need for a trusted third party such as a bank. Since then, the underlying blockchain technology has attracted the attention of various industries. The financial sector expects blockchain to disrupt banking, insurance and asset management. The healthcare industry wants to use the blockchain technology to securely store data. In the energy industry, the blockchain tries to realize different processes in smart grids, which are characterized by a decentralized structure, of which the best-known application is the local P2P energy trading.

As part of the *Green Energy Lab* innovation project[1], companies and research partners in the SGC project proposal are suggesting the research of the use of blockchain technology for the integration of renewable energy into the energy system with a focus on local energy communities. End consumers should be allowed to participate in the electricity market, for which various business models should be evaluated, improved, and simulated. The goal is to implement a platform for local energy trading, based on blockchain technology and to integrate the end users into it using digital consumer devices. Furthermore, on the Smart Grid Controller Blockchain Platform (SGCBP), various applications are to be implemented, e.g., the remuneration of local self-consumption. Based on the results, recommendations for local, regional, and EU-wide political and regulatory frameworks will be developed.

1.2 Problem Statement

This diploma thesis aims to build a blockchain platform as a proof-of-concept which has the main focus on the logging of energy production and consumption inside VPPs, and the settlement of energy surplus and demand between multiple VPPs. The participants of the platform can have various combinations of electrical devices, such as photovoltaic (PV) systems, batteries, electric cars or diesel generators, see Fig. 1. In a real-life environment, participants would be equipped with smart meters or novel Internet-of-Things (IoT) devices that interface with the platform. A central authority, under the control of an energy company or a municipality, would be responsible for the administration of the platform, i.a., responsible for the registration of the smart meters and IoT devices as well as the further processing of the data. The blockchain platform should be extensible so that, based on this, other applications can be realized, e.g., the settlement of punitive duties in case of non-compliance with voltage limits.

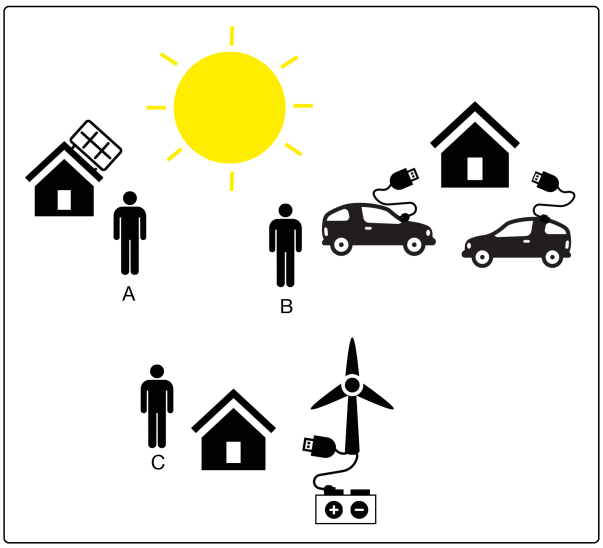

Figure 1: The figure shows an example of a virtual power plant with three participants. Participant A has solar panels on their roof, participant B owns two electric vehicles and participant C has a wind turbine and a battery system.

[1] Green Energy Lab Home - Green Energy Lab `https://www.greenenergylab.at/` (accessed: January 8th 2019)

Choosing a suitable blockchain implementation as the basis for the platform is critical. The blockchain implementation used must fulfill different criteria, e.g., light energy-footprint. Due to the enforcement of the General Data Protection Regulation (GDPR) [Reg16] on May 25th 2018 in the European Union, in this thesis a special focus is put on privacy.

In Austria several energy model regions exist, which target the independence of fossil fuels. In these model regions new concepts and research ideas can be developed and tested. For the proof-of-concept, a model region should be simulated using Raspberry Pis (RPI). These represent all members involved in the operation of the platform. In order to simulate a realistic scenario, defined standard load and generation profiles should be used. Each member should be able to see data according to their role, visualized on a dashboard. For example, these dashboards display energy production and consumption, and transactions in the form of tables and charts.

In addition, the platform deployment process and platform operation in the simulation environment is to be examined with respect to various performance criteria. Depending on the process of creating the platform, as well as the platform deployment and operation evaluation, conclusions are drawn to provide recommendations for the integration of the implemented blockchain platform, for future validation in a real-life environment, e.g., the research and development environment Großschönau.

1.3 Methodology

In a first step, currently active demonstration environments in Austria are presented such as the *Green Energy Lab*, *Act4.energy* or Großschönau, followed by the underlying theory. This theory includes relevant smart grid concepts, privacy, blockchain, hardware and tools. In order to select a suitable blockchain implementation for the platform, criteria have to be deduced from requirements. Based on the selected criteria, a comparison and evaluation of existing blockchain implementations has to be done. The chosen blockchain implementation is then presented.

In a second step, a concept for the blockchain platform is developed. For this, the targeted environment has to be analyzed and data of interest for the platform identified. Based on this a model for the platform has to be described, which includes platform components, business logic and privacy mechanisms. Regarding the simulation, a design is needed that describes the simulated environment and the data used for the simulation.

Then, software packages, such as different Node modules, for the development of the blockchain platform are described. This part is followed by the description of the container technology and tools for the deployment of the platform. Further, the hardware, in this case the Raspberry Pi, is presented.

Next, the implementation of the blockchain platform is described, starting with the porting of the blockchain implementation to the Raspberry Pi and the creation of necessary smart contracts. Then the platform applications, such as dashboards and simulations, are depicted. For the deployment of the platform and its applications onto Raspberry Pis, the platform components and applications have to be built into Docker images. Further, the setup of the operating system (OS) and the required software packages for Raspberry Pis are presented, upon which a microSD image is created for an easier setup. After flashing this image, configuration steps are required. As last step of the implementation, the deployment of the platform and required components are described. This is followed by an examination of the SGCBP, in case of a simulated environment.

Thereby, the deployment of the platform onto the Raspberry Pis, and the platform operation and stability are analyzed.

The last chapter provides a summary of the thesis and an outlook on future blockchain developments. Based on the findings from the implementation and the simulation evaluation, recommendations, and suggestions for the next version of the platform and implementation on hardware in a production environment are given. The chapter concludes with an outlook and vision for smart energy systems regarding blockchain technology.

2 State of the Art and Related Work

In this chapter, first, the demonstration areas *Green Energy Lab*, *Act4.Energy*, and Großschönau, and the Smart Grid Control project proposal are presented. Second, relevant smart grid concepts, such as the virtual power plant concept, are described. Third, the blockchain technology is explained, its basics and the different blockchain types. Further, different research papers and projects with focus on blockchain technology in the energy sector are presented, which range from P2P energy trading platforms to blockchain-prepaid-meters. Finally, relevant hardware and tools for this thesis are characterized.

2.1 Current Demonstration Areas

This section describes the Austrian demonstration areas and model regions *Green Lab Energy*, *Act4.energy* and Großschönau. A model region targets primarily independence of fossil fuels. Thereby, it should focus on local resources to meet energy demands and involve local stakeholders in the development process. This includes utilizing renewable energy sources, increasing energy efficiency, and utilizing new technologies, such as control mechanisms[CF18].

2.1.1 Green Energy Lab

The *Green Energy Lab*[1] is a flagship region of energy in Austria improving the use of green energy technologies with focus on renewable electricity and heat. It includes the four federal states of Burgenland, Lower Austria, Styria and Vienna, thereby reaching around five million energy users, supported by *Energie Burgenland*, *Energie Steiermark*, *Energieversorgung Niederösterreich* and *Wien Energie*. Further, 150 million euro are invested in 31 subprojects, where more than 100 corporate and research partners develop solutions to reach 100 % renewable electricity and heat in Austria.

Individual solutions of research institutions and technology providers from Austria, which have a leading role in the renewable energy and heat sector, will be integrated into solutions for the flexible generation and storage of electricity and heat including the smart involvement of consumers. The project focuses on the innovation fields flexibility, digitalization, integrated systems, customer integration and business models. Subprojects focus, e.g., on technologies such as a blockchain platform for energy exchange, grid-friendly hydropower, and intelligent plug-and-play electricity storage for PV.

The applications for subprojects will be submitted in February 2019 and the *Green Energy Lab* and its subprojects are expected to start in summer 2019.

2.1.2 Act4.energy

The innovation laboratory *Act4.energy*[2] focuses on PV self-consumption optimization and energy stability based on renewable energy. Further, it focuses on the interconnection of electricity, heat and transport as well as e-mobility. The innovation laboratory environment comprises ten communities in southern Burgenland, which already offers a well-developed PV infrastructure. As part of the innovation laboratory, innovative solutions should be developed and tested with the involvement of the population. *Act4.energy* is funded by the *Austrian Research Promotion Agency FFG* as part of the research and technology program *City of Tomorrow*.

2.1.3 Smart Community Großschönau

The municipality Großschönau in Lower Austria's Waldviertel has been dealing with topics in the energy sector since the 1980s. In 2011 the research center for construction and energy in Großschönau opened, which is located in the first European passiv house village[3]. Until 2017 it enabled interested guest to test passiv houses for two to seven days[4]. The project *Municipality Großschönau as virtual energy storage (GAVE)* analyzed consumer energy management, especially automated electric load management. It focused not only on the implementation of necessary technologies but further on the acceptance by the local residents[5].

2.2 Relevant Smart Grid Concepts

In this section, the smart grid concept and virtual powerplant concept, their development, and several definitions are presented. This section concludes with a description of the Smart Grid Control project proposal.

2.2.1 Smart Grid

The integration of renewable energy sources, the need for demand response, and energy conservation are just some challenges, which today's energy systems are confronted with. The traditional electricity grids carried the power unidirectionally from power plants to the customers. Due to the unidirectional information flow, gathering information about the grid in real-time was not possible. To avoid blackouts, grids had to be built to withstand rarely occurring maximum demand peaks. This lead to an inefficient utilization of the grid, which is still present today with many opportunities to increase the hosting capacity [Far10].

[2]Detailansicht - Energie Kompass GmbH `https://www.act4.energy/nc/de/projekt/projektdetail/innovationslabor-act4energy/` (accessed: January 8th 2019)

[3]Sonnenplatz Großschönau `http://www.sonnenplatz.at/page.asp/lang%3Den/Forschung-Projekte` (accessed: January 8th 2019)

[4]Sonnenplatz Großschönau - 1. European Passivhausdorf `http://www.sonnenplatz.at/page.asp/lang%3Den/passivhausdorf` (accessed: January 8th 2019)

[5]GAVE – Municipality Großschönau as virtual Energy Storage – ICT Energy&IT Group `https://energyit.ict.tuwien.ac.at/projects/gave` (accessed: January 8th 2019)

The biggest problem for utilities was the distribution network, which was like a blackbox to them. Without information from the distribution network and control methods, generation capacities could not be efficiently expanded to satisfy the rising electricity demand. Therefore, solutions have to be developed, to further integrate information and communication technology into the grid. Especially the distribution network and the possibility for demand-side management (DSM) need to be extended [Far10].

Automated meter reading (AMR) in distribution networks allow to remotely read consumption and status information but do not include any control over the devices or DSM. The next step was the development of the advanced metering infrastructure (AMI), which utilized a two-way communication. On one hand, it enabled to remotely gather information about the network and demand. On the other hand, it allowed to react based on the collected information and set actions in the distribution network. The AMI serves as basis for the development of the smart grid [Far10].

Due to different focuses in research, a unique definition on the term "smart grid" has not yet surfaced [FMXY12]. The *Electric Power Research Institute (EPRI)* [VD+09] defines the smart grid as following: "The term "Smart Grid" refers to a modernization of the electricity delivery system so it monitors, protects, and automatically optimizes the operation of its interconnected elements – from the central and distributed generator through the high-voltage transmission network and the distribution system, to industrial users and building automation systems, to energy storage installations, and to end-use consumers and their thermostats, electric vehicles, appliances, and other household devices."

A very early definition of smart grids was found by the members of the *Technology Platform Smart Grids Austria (TPSGA)*: "Smart Grids are power grids, with a coordinated management, based on bi-directional communication between grid components, generators, energy storages and consumers to enable an energy-efficient and cost-effective system operation that is ready for future challenges of the energy system." [LFP08]

By using bidirectional flow of electricity and information in every level of the grid, the smart grid should be able to provide quality, reliability, and efficiency by dynamic optimization. As a self-healing system, it should be able to respond to events, e.g., failures or demand peaks, irrespective of their location. While supporting all energy generation and storage possibilities, consumer involvement in the operation of the grid and energy market should be enhanced. Instead of instantaneously replacing the existing grid, the smart grid should be developed parallel to the existing electricity grid and gradually replace it [Far10].

2.2.2 Virtual Power Plant

Increase of decentralized energy resources (DER) introduces new challenges to power systems. First, most DER units, e.g., PVs, are weather-dependent and therefore intermittent, which means that their output is fluctuating and only partially controllable. Second, the participation of DERs on the energy market is aggravated due to the small size and intermittency. Finally, most DER units are operated isolated to only satisfy local needs, without contributing to the grid. To overcome these challenges, multiple DER units are aggregated and actively controlled in a VPP [PRS07].

Still, there does not exist a unique definition for VPPs. Generally, a VPP can be described as in [PDVBD15]: "A portfolio of DERs, which are connected by a control system based on

information and communication technology (ICT). The VPP acts as a single visible entity in the power system, is always grid-tied and can be either static or dynamic." This definition includes the following DERs: distributed generation units, energy storage units, and flexible loads. Due to the aggregation, the VPP can be characterized like a transmission-connected generating plant, by parameters, e.g., the scheduled output, output limits, or reserve capacity.

An Energy Management System (EMS) [SMT11] controls the generation units, storage units, and controllable loads according to specific targets, such as the maximization of profits. Therefore, a bidirectional communication is necessary. The EMS has to be able to receive information, like generatio forecasts, from all units, and send commands to them. It can be realized as a centralized system with a control center or as a decentralized system, where every unit is controlled by a separate local controller or as a mixed system.

Derived from the VPP's main activities, the participation in the energy market and the transmission system management, two different VPP roles exist, the commercial VPP (CVPP) and the technical VPP (TVPP).

The CVPPs [SMT11] represent DERs, which can be located in several distribution and transmission grids, in energy markets as one generating power plant, connected through the transmission grid. Therefore, it has a profile, aggregated from operating and cost characteristics of each DER unit. Its tasks are energy trading, optimization and scheduling of energy production depending on forecasted demand and passing information about the DER units to the TVPP. The CVPP facilitates the visibility and participation of each DER unit in the energy market. Further, the operation of the network is not of concern for the CVPP.

The TVPP [SMT11] represents DERs, which are located in the same network, to system operators. It is responsible for distribution system operator's (DSO) system management and to provide services like system balancing to the transmission system operator (TSO) by optimally using the capacity of the DER units. Therefore, the TVPP has to receive information, e.g., operating and cost parameters, about the local network and the DER units from the representing CVPPs and aggregate these. Often DSOs act as TVPP, thereby optimizing the operation of their network by using the DER units.

2.2.3 Related Smart Grid Control Approaches

In demonstration areas, such as the *Green Energy Lab*, new concepts and research ideas for smart energy systems are being developed and tested. The Smart Grid Control project is a possible subproject for the *Green Energy Lab*. It aims at increasing the integration of renewable energy generation into the energy system with a focus on the local market and customer involvement. Thereby, different market platform concepts will be evaluated and a platform for local energy exchange trading based on blockchain will be developed, which will assist the integration of renewable energy sources by, e.g., valorising ancillary services using a hybrid battery energy storage system. For the implementation of the platform the integration of the end-consumers with digital end-user devices, such as smart meters, is crucial. Further, RES, storages, flexible demand and other energy system technologies will be combined into commercially viable products, thereby developing new business models, which enhance the involvement of prosumers and consumers. Based on the assessment, simulation and evaluation of these business models and the platform, recommendations and proposals for local, regional and EU-wide policy and regulatory frameworks will be given.

2.3 Privacy

In this section, first, the General Data Protection Regulation and its consequences are presented. This regulation has been enforced in the European Union (EU) on May 25th 2018. Thereby, privacy by design has been implemented as legal requirement. Therefore, this section describes privacy design strategies and provides examples for these.

2.3.1 General Data Protection Regulation

Due to the increasing number of privacy scandals, like the *Facebook Cambridge Analytica data scandal*[6], and emerging technologies, like cloud storage and IoT, a detailed regulation of privacy became crucial. While the Data Protection Directive, Directive 95/46/CE [C+95], was adopted in 1995 in the European Union, to unify data privacy laws of the member states. As directive it left space for interpretation. Since 1995, the number of data-driven applications extremely increased, e.g., the evolution of social media, and cases regarding privacy, e.g., the invalidation of the Safe Harbour [EU15], were raised. Therefore, the regulatory environment of the European Union had to be updated.

On April 14th 2016 the EU Parliament approved the General Data Protection Regulation [Reg16], which replaces the Directive 95/46/CE. GDPR was enforced in EU member states on May 25th 2018, thereby establishing a unique privacy law in the EU to fulfill their aim of protection of EU citizens' data privacy.

Any information, which can be used to identify a person like name and email address, is considered as personal data. Companies, regardless of their location, which process personal data of data subjects residing in the EU have to comply to GDPR. GDPR does not refer only to data processing entities, called data processors, but further to entities on whose behalf the data is processed, called data controllers. Organizations violating GDPR, are fined according to a tiered approach, with fines going up to 4 % of the annual global turnover or 20 million euro minimum.

Besides strengthening conditions for consent, e.g., by requiring intelligible requests for consent, data subjects have been entitled with the *Right to Access* and *Right to be Forgotten*. While the *Right to Access* gives data subjects the right to obtain information on their personal data from data controllers, e.g., which data is being processed, the *Right to be Forgotten* among other things authorizes the data subject to request the erasure of its personal data by the data controller. Further, GDPR demands notification within 72 hours in case of data breaches.

Especially, due to the implementation of privacy by design as legal requirement, data controllers and data processors have to change their existing systems and build new systems accordingly to it, by using techniques like data minimization and anonymization.

2.3.2 Privacy Design Strategies

GDPR specifies privacy by design as legal requirement. For privacy by design, eight privacy design strategies are presented in [Hoe14]. These privacy design strategies can be grouped into two categories. The first category is comprised of data-oriented strategies as:

[6]The Cambridge Analytica Files `https://www.theguardian.com/news/series/cambridge-analytica-files` (accessed: January 8th 2019)

- Minimise: Only necessary data should be collected, thereby minimizing the amount of processed personal data. An example is *select before you collect*, where instead of collecting all available data and afterwards selecting which data is necessary, the necessary data is first selected and then only this necessary data is collected.
- Hide: By hiding personal data and their interrelationships from plain view, unlinkability and unobservability should be achieved. Encryption, anonymization and pseudonyms fall under this strategy.
- Separate: Personal data should not be processed centrally, but rather distributedly. This means, that data from separate sources should be stored separately and these storages should not be linkable.
- Aggregate: To reduce the amount of detail and thereby the data sensitivity, data should be aggregated before being processed. In context of energy systems, this can be achieved by logging energy production or consumption in intervals.

Besides data-oriented strategies, there exist process-oriented strategies, which are:

- Inform: Regarding transparency, informing data subjects about the processing of their personal data is crucial. The data subject should have access to information, such as the purpose of the processing, the processing parties, and data access rights. An example are notifications in case of data breaches, as demanded by the GDPR.
- Control: Complementary to the inform strategy, the control strategy allows the user to control the use of their personal data. Privacy settings in applications, which can be changed by the user, belong to this strategy.
- Enforce: If a privacy policy exists for the system, the policy should be enforced by implementing mechanisms to prevent policy violations. An example for this strategy is access control.
- Demonstrate: Data controllers should be able to give disclosure about their implementation of privacy policies. For this purpose, logging and auditing systems can be used.

Rather than only using one strategy, multiple of these strategies should be applied together. For example, privacy enhancing technologies (PET) try to combine multiple privacy strategies to achieve goals, such as transparency and data minimization.

2.4 Blockchain

This section first explains the basics of the blockchain technology and the distributed ledger technology. Then the different types of blockchain are listed and described. Essential for the blockchain is the consensus mechanism, of which various versions are described here. In order to be able to select one blockchain implementation for the thesis, different blockchain implementations are described and compared according to certain criteria. Based on the comparison, one blockchain implementation is selected.

2.4.1 Basics

The blockchain technology was first introduced in 2008 in the Bitcoin white paper [Nak08], with the aim to establish a peer-to-peer currency without the need for a trusted third party, e.g., banks. Since then different use cases of the blockchain have been researched and realized, e.g., a healthcare record system.

The blockchain can be described as a distributed ledger, which stores records in blocks. These blocks are chained together by referencing the previous block using cryptographic hashes, as shown in Figure 2. This leads to a tamper-proof ledger. Every node in the blockchain network has a copy of the blockchain and a public/private key pair. The public key is used to address the node and the private key to sign transactions. When making a transaction, it is first signed and then broadcasted to the neighboring nodes. These nodes then validate the transaction and forward it only if it is valid. Thereby validated transactions get broadcasted to the network and collected by the nodes. Repetitively nodes group received validated transactions into a block to be added to the blockchain. These blocks can differ from each other, e.g., due to propagation of the transactions through the network, which leads to different copies of the blockchain in the network and different views on the current state [TS16].

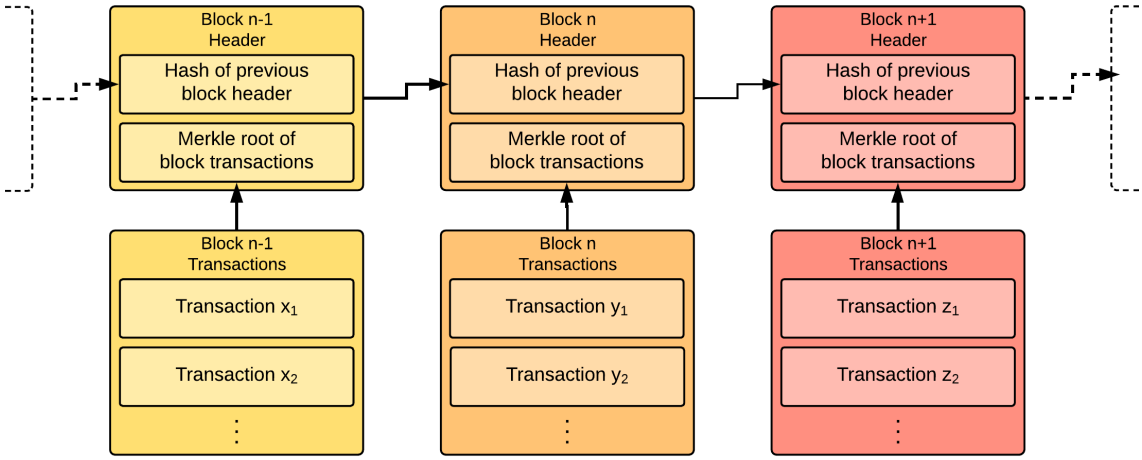

Figure 2: The figure shows a visualization of the generic blockchain data structure. A block contains the hash of the previous block header and a merkle root. The merkle root represents the block's transactions.

Therefore, a consensus mechanism is needed to achieve a common view within the network. Multiple consensus mechanisms with different approaches exist. Which one to choose depends primarily on the blockchain type. Public networks mainly adopt Proof-of-Work (PoW) or Proof-of-Stake (PoS) as consensus mechanism. PoW, which is used by Bitcoin, is solving cryptographic puzzles by brute-forced cryptographic hashing. The node, which solves this puzzle, is rewarded and its block is added to the blockchain. The biggest point of criticism of PoW is its high energy consumption due to the extensive cryptographic hashing. In contrast, in PoS, where the node's chance to create the next block is proportional to its balance, the number of needed computations is reduced. Most of the consensus mechanisms used in private networks, where the participants are known, are based on solving the Byzantine Generals Problem, whereby the most prominent one is the Practical Byzantine Fault Tolerance (PBFT) algorithm [TS16].

The blockchain enables participants to securely interact in a distributed P2P system without having to trust each other. Further, the network is able to withstand node faults and to agree on a common view. Additionally, it is tamper-proof, transparent, and verifiable. Combined with Internet-of-Things (IoT), the blockchain could transform several industries and create new business models and applications [CD16].

Smart contracts were first mentioned in 1994 in [Sza94] by Nick Szabo with the vision to realize legal contracts as computer code. With the upcoming of the blockchain technology, smart contracts gained interest too. Running atop of the blockchain, smart contracts can be used to reach and enforce agreements automatically according to specified rules, such as energy trading based on a smart contract. The use of smart contracts leads to cost reduction, especially for low-value transactions, and trust-free agreements[7].

2.4.2 Blockchain Types

For different use cases and requirements some blockchain types are more suitable than others, e.g., for the financial industry primarily private blockchains are being researched. Still, a consensus on the classification of blockchain types does not exist[8].

One approach is to differentiate blockchains into public, private, and consortium blockchains. In public blockchains everyone can send transactions, read transactions, and participate in the consensus process. Consortium blockchains are under control of a group, e.g., a group of banks. The consensus process is restricted to selected participants, and the permission to read can be restricted too. Compared to public blockchains, consortium blockchains are faster and offer more privacy. In private blockchains the permission to write is restricted to one entity, e.g., an organization. Further, the permission to read may be restricted[8].

Another approach is to differentiate blockchains into permissioned and permissionless blockchains. In permissionless blockchains the user permissions to read data from and write data to the ledger are not restricted. Further, in reaching consensus each member can participate. While permissionless blockchains deliver transparency and a trust-free environment, these are often slower than permissioned blockchains due to consensus mechanisms like PoW. In contrast, permissioned blockchains are owned by some entity and therefore reaching consensus is carried out by trusted members. Due to its private membership permissioned blockchains are faster than permissionless blockchains and of particularly interest for banks[9].

The distinction between the presented approaches is not always clear and sometimes they are treated as the same, e.g., public blockchains are often seen equal to permissionless blockchains and private blockchains to permissioned blockchains[9].

2.4.3 Privacy & Blockchain

Due to the power grid as critical system and laws, such as the GDPR in the European Union, privacy and data protection have to be taken into account. In public blockchains, the transaction data is publicly stored at every node which raises privacy issues. Although Bitcoin uses

[7] Smart Contracts `https://blockchainhub.net/smart-contracts/` (accessed: January 8th 2019)

[8] Blockchains & Distributed Ledger Technologies `https://blockchainhub.net/blockchains-and-distributed-ledger-technologies-in-general/` (accessed: January 8th 2019)

[9] Private Vs. Public and Permissioned Vs. Permission-less `http://blocktonite.com/2017/06/27/private-vs-public-and-permissioned-vs-permission-less/` (accessed: January 8th 2019)

pseudonymization, through statistical analysis de-anonymization can be achieved. Therefore, solutions are needed to ensure privacy and data protection by design of the blockchain [HP17].

To ensure, that only involved participants can read the transaction data, while still being able to verify transactions and ensure that the transaction data exists, different cryptographic techniques are used. The simplest approach would be to only store encrypted data on the blockchain[10]. Thereby, only participants with an encryption key would be able to decrypt the data. Even if the data is encrypted, metadata could be used for statistical analysis[11].

Another approach is to mix transactions from different users, thereby obscuring the origin of payments. Due to the dependence on a third party, called mixer, a decentralized approach, called CoinJoin, was proposed. Still mixers are not a suitable solution if the privacy set, the number of involved participants in the mixing, is too small, because de-anonymization is possible. Therefore, it would be needed that every user participates in the mixing[12].

State channels are a solution, which additionally tackles scalability issues. These are off-chain and only known to its participants. These allow to perform some operation off-chain and when closing them to put the result on-chain. For example, state channels can be used for micro-payments, where when closing the channel, micro-payments are aggregated and with one transaction put on-chain[13].

Confidential transactions enable to obscure transaction amounts and balances. Thereby, only participants involved in the transaction can see the transferred amount. The use cases of these confidential transactions are limited[13].

Ring signatures do not obscure the transferred amount but disguise the connection between sender and receiver. Thus, participants form a group, called ring, enabling every participant to sign transactions on behalf of the group, without revealing which participant signed it. A restricting factor is the number of involved participants, called ring size, whereby a small ring size is more prone to de-anonymization[12].

The most promising technology are zero knowledge proofs (ZKP), which allows parties to prove a proposition, e.g., change of state, without revealing any other information. A non-interactive type of ZKPs are zk-SNARKs (Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge). While the proofs are small and can be verified fast, their generation requires resource intensive computation[13]. Due to this, zk-SNARKs are not suitable for less powerful devices and still show scalability issues. Another drawback is the setup phase, which is needed for secret generation. On one hand, for every Turing-complete smart contract a new setup phase is required. On the other hand, if the setup phase is compromised, the security cannot be maintained[14].

Applications already using zk-SNARKs are the Bitcoin protocol extension Zcash, and the smart contract system Hawk. In Zcash the Bitcoin transaction format has been changed to integrate encrypted transactions, where the data is encrypted by using zk-SNARKS and only authorized

[10] Differentiating Between Privacy and Secrecy on the Blockchain `https://bitcoinmagazine.com/articles/differentiating-between-privacy-and-secrecy-blockchain/` (accessed: January 8th 2019)

[11] Privacy on the Blockchain `https://blog.ethereum.org/2016/01/15/privacy-on-the-blockchain/` (accessed: January 8th 2019)

[12] Mixers and ring signatures `https://blog.keep.network/mixers-and-ring-signatures-51f3f125485b` (accessed: January 8th 2019)

[13] The Trend Towards Blockchain Privacy: Zero Knowledge Proofs `http://sammantics.com/blog/2016/8/23/the-trend-towards-privacy-how-blockchains-plan-to-accomplish-this` (accessed: January 8th 2019)

[14] Zero-knowledge proofs, Zcash, and Ethereum `https://blog.keep.network/zero-knowledge-proofs-zcash-and-ethereum-f6d89fa7cba8` (accessed: January 8th 2019)

participants are able to see it. Thereby, users can perform private transactions, where sender, receiver and transferred amount can be obscured. Hawk offers private smart contracts, thereby obscuring the code of the smart contract and any data sent to and received from the smart contract, e.g., transferred amounts. To not put a strain on the network, resource-intensive computations in Hawk are done off-chain[13].

2.4.4 Blockchain Projects For Energy

The use of the blockchain technology in the energy system related areas is currently being extensively researched. The most prominent use case of the blockchain technology is P2P energy trading, where energy trading between two parties is processed directly via the blockchain, without the involvement of third parties, as can be compared with [SW17] and [MPM17]. Researched topics related to P2P energy trading are further transactive energy auctions in [HSLC17], electricity trading with digital-grid routers [TNA17] and energy trading for industrial IoT [LKY+18]. In [SDSG+17] and [DSGI+18] blockchain is used to track energy losses and thereby improving attribution of these. The blockchain-based green certificate market presented in [CCMN17] and [MMM17] introduces a blockchain platform for distributed optimization and control of energy resources in microgrids. The use of blockchain technology for micro-payments is of special interest for IoT [LBA17], which could be an important part in building smart cities, e.g., in the smart district model in [LR17]. Furthermore, there are different use cases of blockchain for electric vehicles (EV), such as EV-P2P trading in smart grids [KYH+17], EV and charging pile management system [HXWL18], or an autonomous selection of an EV charging station [PKS16].

Some projects have been already developed or are at time of writing object of research. A technical overview of projects can be found in [GMF+17] and [PMG+18]. Most of them are based on scenarios with a focus on peer-to-peer energy trading. With Brooklyn Microgrid, a community energy market was established, in which residents of the Brooklyn neighbourhood can trade energy with each other. Similarly, PWR.Company enables P2P trading in microgrids by equipping the participants with deep cycle batteries. With WePower, a blockchain-based green energy trading platform, energy producers can issue own energy tokens, which ensure that the producer delivers the energy to the buyer. The Key2Energy concept describes tenant electricity platforms where tenants can buy energy directly from self-generating residential buildings at cheaper prices. Share & Charge has already developed a charging stations network for electric vehicles, including public and private charging stations. Dajie produces IoT devices, which are used in combination with a blockchain-based platform for P2P energy exchange and to pay energy and services to energy companies. Wholesale P2P energy trading with blockchain technology is provided by PONTON Enerchain.

Other projects aim to increase the use of renewable energy sources. With the help of blockchain, the production and consumption of renewable energy are logged and rewarded accordingly. NRGcoin rewards producers of renewable energy per kWh if the production is in line with local demand. GrünStromJeton reviews the electricity mix used and rewards the consumers when renewable energy sources are used. SolarCoin remunerates PV system owners per MWh produced to reduce long payback times.

Some projects have a rather specific focus, like Bankymoon and TheSunExchange. Bankymoon develops blockchain-aware prepaid meters that can be recharged remotely by using cryptocurrencies. These are of particular interest to African countries that already use prepaid meters. TheSunExchange offers blockchain-crowdfunding, through which solar cells are purchased and

then leased. This facilitates the electrification of rural, less industrialized areas without access to the grid.

Particular of interest for system operators are Electron, PONTON's Gridchain and PowerLedger. Electron developed a meter registration platform, which enables switching the energy-supplier in near real-time, and an exchange platform for demand-side response actions. PONTON's Gridchain focuses on processes for the real-time grid and PowerLedger provides market trading and clearing mechanisms.

All-in-one platforms are developed by GridSingularity and LO3 Energy. GridSingularity wants to create a distributed data exchange platform based on an agent, which serves as the basis for a wide range of applications, from P2P trading to registration platforms. LO3 Energy's Transactive Grid can be described as a blockchain-based microgrid intelligence system consisting of TransActive Grid elements (TAG-e). It targets various business models for the distributed grid and the transactive energy space, such as P2P energy transactions and the control of DERs for grid balancing.

Further, the Energy Web Foundation (EWF), a non-profit organization, whose goal is to accelerate the use of blockchain technology in the energy sector, is implementing various activities:

- development of an open-source IT infrastructure
- development of an EWF Blockchain
- analysis of different use cases
- development of use cases for proof-of-concepts and commercial applications

2.5 Hardware and Tools

In this section the used hardware, which is the Raspberry Pi, and tools are described. These tools comprise software packages and encryption technologies.

2.5.1 Hyperledger Fabric SDK

To interact with a Hyperledger Fabric network the Hyperledger Fabric SDK can be used. Until October 2018, the SDK has been released for Node.js[15] and Java[16] and is still in development for Python, REST and Go. The SDK consists of the three modules api, fabric-client and fabric-ca-client. Developers can implement their own versions of the SDK's key interfaces by using the module api, e.g., own cryptographic algorithms. The module fabric-client enables applications to interact with the HLF network in multiple ways to[15]:

"• create channels,
- ask peer nodes to join the channel,
- install chaincodes in peers,

[15]Hyperledger Fabric SDK for node.js Index `https://fabric-sdk-node.github.io/` (accessed: January 8th 2019)

[16]Java SDK for Hyperledger Fabric 1.3 `https://github.com/hyperledger/fabric-sdk-java` (accessed: January 8th 2019)

- instantiate chaincodes in a channel,
- invoke transactions by calling the chaincode, and
- query the ledger for transactions or blocks."

The SDK requires user certificates signed by a CA for signing requests made to the HLF network. Therefore, the module fabric-ca-client offers functions to register and enroll a user in order to retrieve the user's enrollment certificate[15].

2.5.2 Node.js

Node.js[17,18] is a "production grade" asynchronous JavaScript runtime, for server-side network applications. Due to its event-based model and input/output methods, which are available as asynchronous non-blocking functions, Node.js is suitable for scalable systems and real-time applications. Further, modules for Node.js can be installed by using the package manager npm[19]. There exists a broad range of modules including modules for networking, file system operations and cryptographic functions. Some organizations, which use Node.js for their applications, are PayPal, Netflix, eBay, or NASA[20].

2.5.3 React

React[21,22] is a JavaScript library for creating user interfaces, which was developed by Facebook and open-sourced in May 2013 to the public. The user interface is composed of independent and reusable components. When developing a React application, it has to be taken into account that the data flow between the components is unidirectional[23]. The implemented reconciliation algorithm[24], which is used in React's virtual DOM, improves the speed of re-rendering components on updates. Therefore, it is suitable for high-performance applications. Based on React, Facebook released in March 2015 React Native[25], a library for the development of native mobile applications[26].

2.5.4 Advanced Encryption Standard

Due to its short key length of 56 bits, the Data Encryption Standard (DES) became vulnerable to brute-force-attacks and inadequate for protecting sensitive information. Therefore, the

[17] Node.js `https://nodejs.org/en/` (accessed: January 8th 2019)
[18] nodejs/node: Node.js JavaScript runtime `https://github.com/nodejs/node` (accessed: January 8th 2019)
[19] npm `https://www.npmjs.com/` (accessed: January 8th 2019)
[20] 10 best Node.js app examples from Uber to NASA `https://thinkmobiles.com/blog/node-js-app-examples/` (accessed: January 8th 2019)
[21] React – A JavaScript library for building user interface `https://reactjs.org/` (accessed: January 8th 2019)
[22] A declarative, efficient, and flexible JavaScript library for building user interfaces. `https://github.com/facebook/react` (accessed: January 8th 2019)
[23] Thinking in React – React `https://reactjs.org/docs/thinking-in-react.html` (accessed: January 8th 2019)
[24] Reconciliation – React `https://reactjs.org/docs/reconciliation.html` (accessed: January 8th 2019)
[25] React Native: A framework for building native apps using React `https://facebook.github.io/react-native/` (accessed: January 8th 2019)
[26] A brief history of React Native – React Native Development – Medium `https://medium.com/react-native-development/a-brief-history-of-react-native-aae11f4ca39` (accessed: January 8th 2019)

National Institute of Standards and Technology (NIST) decided to develop the Advanced Encryption Standard (AES). As part of the development, a call for algorithms was issued on September 12th 1997[27]. Thereby, the Rijndael algorithm for a block length of 128-bit was selected as AES algorithm. It can be used with three different cipher key lengths: 128, 192 and 256 bits. Deduced from this, the different forms of the algorithms are named AES-128, AES-192 and AES-256 [Sta01].

Multiple attacks on the AES algorithm are known, like [Rij10] and [BKR11]. Nevertheless, as of October 2018, besides side-channel attacks, which do not attack the algorithm but its implementations, only impractical attacks for a correctly implemented AES implementation are known to the public. An example for such a side-channel attack, which is based on monitoring the cache, and its application on OpenSSL are described in [OST06].

2.5.5 Cipher Block Chaining

Block cipher algorithms, such as AES, encrypt only one fixed-sized block. For the encryption of multiple blocks, a mode of operation is needed. The mode defines how to apply the block cipher to multiple blocks. The Cipher Block Chaining (CBC) mode was invented by Ehrsam, Meyer, Smith and Tuchman in 1976 [EMST76]. In the CBC mode, each plaintext block is XORed with the previous ciphertext block and then encrypted. For the first plaintext block, an initialization vector (IV) for XORing is required. Due to the use of different IVs, the encryption of the same plaintext with the same key produces different ciphertexts. The IV does not need to be kept secret, but it should be unpredictable and not reused with the same key. Otherwise an attacker could gather necessary information for an attack. Because the IV is necessary for decrypting, the IV is usually stored together with the ciphertext. Due to fixed block sizes and variable plaintext lengths, the last plaintext block has to be padded before encryption, if it is smaller than the fixed block size [Dwo01].

2.5.6 Containers

Containers are similar to virtual machines (VM), both enable to package applications and dependencies, e.g., libraries and runtimes. Thereby the applications are run in an isolated environment. In contrast to heavy-weight VMs, where the hardware is virtualized and atop the host operating system another OS has to be run, the light-weight containers virtualize resources at OS-level (see Figure 3). The advantages of containers are their capability to run independently of the underlying system, which requires less memory than VMs, and start faster. Further, multiple containers running on the same system, are isolated from each other, which eliminates dependency and resource conflicts[28,29].

[27] AES Development `https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development` (accessed: December 15th 2018)

[28] Get Started, Part 1: Orientation and setup `https://docs.docker.com/get-started/` (accessed: January 8th 2019)

[29] What are containers and their benefits — Google Cloud `https://cloud.google.com/containers` (accessed: January 8th 2019)

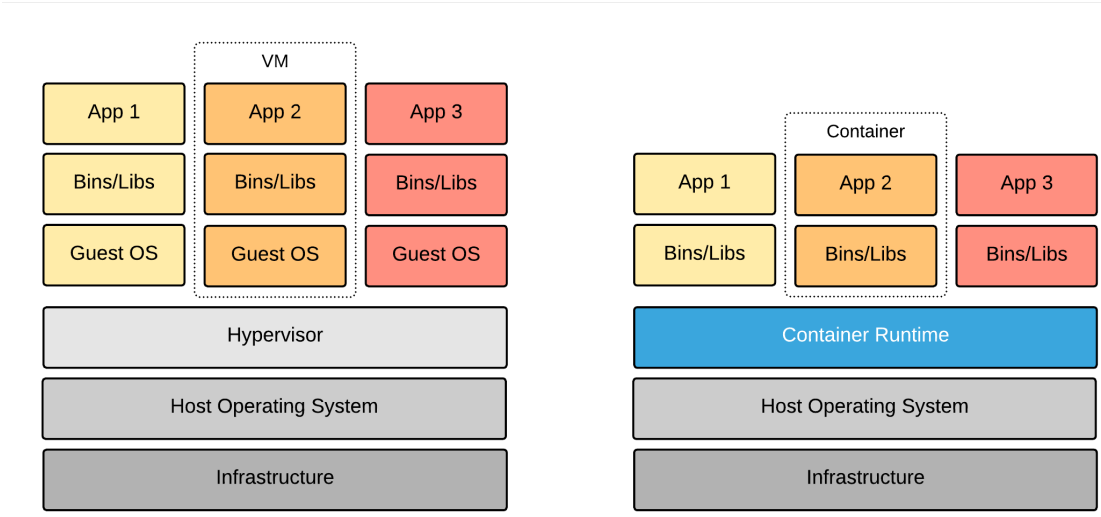

Figure 3: The figure shows the architecture of virtual machines and containers. (based on 29)

Due to the predictability of the container environment and independence from the target environment, the development of applications can be improved. Because of the container's interchangeability, which enables to update and upgrade applications, and portability, which enables containers to be deployed to any environment, the development of applications is decoupled from their deployment. Additionally, containers are highly scalable and transparent, which allows to monitor and manage applications. To create container images different container formats can be used, such as the Docker container format[29].

2.5.7 Raspberry Pi

In 2006 at the University of Cambridge, the idea of a small cheap computer for schools and universities to promote computer sciences was born. To realize such a device, the Raspberry Pi Foundation was founded. The Raspberry Pi 1 Model A and Model B were created and released to the market in 2012[30]. Since then various models, with support for Ethernet, USB hubs and Wireless LAN, and components, e.g., camera modules, have been created. Thereby, extending the number of possible use cases of the Raspberry Pis. For example, a Raspberry Pi can be used as office computer, as network attached storage or for home automation[31].

For the operation of the Raspberry Pis an operating system is needed, which is played on a SD card or microSD card. Several operating systems are available, such as the officially recommended Raspbian, which is based on Debian. Furthermore, operating systems that are optimized for certain requirements exist, e.g., OpenWrt, which is optimized for routing network traffic[30].

[30] RPi General History `https://www.elinux.org/RPi_General_History` (accessed: January 8th 2019)
[31] Raspberry Pi: Top 35 projects to try yourself `http://www.itpro.co.uk/mobile/21862/raspberry-pi-top-31-projects-to-try-yourself-1/page/0/1` (accessed: January 8th 2019)

2.5.7.1 Raspberry Pi Zero W

The Raspberry Pi Zero W[32], shown in Figure 4, is the smallest computer in the Raspberry Pi family. It has a single-core CPU with a clock speed of 1 GHz, 512 MB RAM, a micro USB power supply and the interfaces, like Mini HDMI and USB OTG. For the simulation, the Raspberry Pi Zero W is used, which was introduced in February 2017 as an extended Raspberry Pi Zero. Unlike the Raspberry Pi Zero, the Raspberry Pi Zero W has advanced connectivity options such as 802.11 b/g/n Wi-Fi, Bluetooth 4.1 and Bluetooth Low Energy.

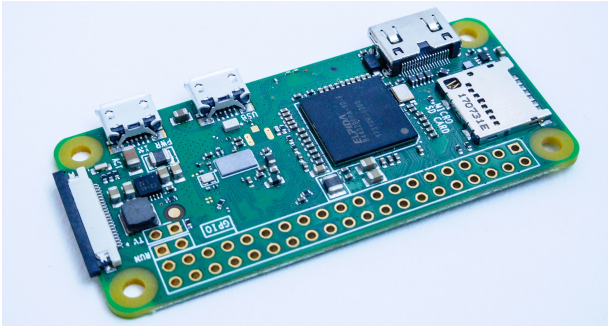

Figure 4: The figure shows a picture of the Raspberry Pi Zero W

2.5.7.2 Raspberry Pi 3 Model B

The Raspberry Pi 3 Model B (RPI3B)[33], shown in Figure 5, was introduced in February 2016 as successor of the Raspberry Pi 2 Model B (RPI2B). It has a quad-core CPU with a clock speed of 1.2 GHz, 1 GB RAM, and different interfaces, like one Ethernet, one HDMI and four USB ports. Compared to the RPI2B, the RPI3B has advanced connectivity options such as 802.11 b/g/n Wi-Fi, Bluetooth 4.1 and Bluetooth Low Energy already built-in. Due to the RPI3B's ARMv7 architecture, it supports 64-bit operating systems.

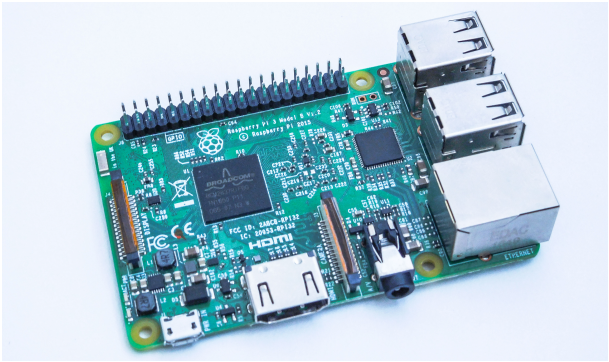

Figure 5: The figure shows a picture of the Raspberry Pi 3 model B

[32]Raspberry Pi Zero W `https://www.raspberrypi.org/products/raspberry-pi-zero-w/` (accessed: January 8th 2019)

[33]Raspberry Pi 3 Model B `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/` (accessed: January 8th 2019)

3 Selection Of Hyperledger Fabric

For selection of an appropriate blockchain implementation for the SGCBP, multiple blockchain implementations are introduced and compared in this chapter. Then the selection of Hyperledger Fabric as blockchain implementation for the SGCBP is argued. Furthermore, this chapter describes Hyperledger Fabric and its architecture, by starting with the different nodes, the ledger and the channels. Then the Membership Service Provider component, chaincodes and transaction workow are presented. The information on Hyperledger Fabric is taken from the official HLF v1.1 documentation [Hyp18].

3.1 Evaluation Of Blockchain Implementations

In this section, first the requirements for the selection of a blockchain implementation, which acts as basis of the blockchain platform, are described. Second, different blockchain implementations and their features are presented. Last, the presented blockchain implementations are compared and one is selected for the implementation of the blockchain platform.

3.1.1 Blockchain Requirements

Besides increasing the share of RES, the EU aims to increase energy efficiency by at least 27 %. With regard to this, the use of the extremely energy-consuming consensus mechanism Proof-of-Work, as for Bitcoin, is not suitable. The platform should take this into account and use a consensus mechanism with a light energy-footprint.

It should be possible to integrate smart meters and IoT devices into the platform. Their resources, such as processor power and memory space, are limited. Therefore, the selected blockchain implementation needs to be resource-aware.

Although, the blockchain enables multiple parties to interact securely in a distributed P2P system without trusting each other, the power grid is a highly-critical infrastructure and should not be accessible to everybody. Therefore, the access to the platform has to be restricted. Only selected participants should be able to read from and write to the ledger.

The platform has to process the energy production and consumption data of the participants, which is by definition personal data. This requires privacy mechanisms, such as confidential

transactions and encryption for the platform. Therefore, a blockchain implementation, already integrating privacy mechanisms is necessary.

Processes, such as logging of energy production and consumption and the settlement of these, are to be integrated into the platform. To integrate these functionalities, the blockchain implementation has to support the integration of a business logic, e.g., with smart contracts.

Furthermore, for optimization purposes, the blockchain implementation should be modular and offer possibilities to integrate other components, e.g., a new consensus algorithm. This modularity would facilitate, e.g., upgrading components of the blockchain implementation to newer versions.

In respect of these requirements, following criteria have been chosen for the selection of a blockchain implementation for the SGCBP:

- Light energy-footprint
- Modular
- Permissioned
- Privacy mechanisms
- Resource-aware
- Smart contracts

3.1.2 Blockchain Implementations

On the following pages, the blockchain implementations Ethereum, Quorum, Hyperledger Fabric, Hyperledger Sawtooth and Corda are described. The descriptions focus on their architecture, features, and privacy mechanisms, having the requirements in mind.

3.1.2.1 Ethereum

Ethereum developed as a blockchain platform with the ability to be reprogrammable is available as an open-source project. It does not focus on any specific field, and therefore is of interest for a variety of use cases[34].

As a permissionless public blockchain, it allows everybody to participate in the network. Every Ethereum node runs the Ethereum Virtual Machine (EVM) and the nodes communicate via a P2P network protocol. The blockchain tracks the state of every account, which are public/private key pairs and serve as identity in the blockchain network[34].

Smart contracts in Ethereum are comparable to an autonomous agent and run in the EVM. A smart contract can read and write data, perform computations, or call other contracts. They can be written in the programming languages Solidity, Serpent, and LLL[34].

Currently Ethereum uses Proof-of-Work as consensus algorithm. Participants, called miners, have to solve cryptographic puzzles and create new blocks, which are then verified by other participants[34]. The biggest drawback of PoW is its high energy consumption. Therefore, Ethereum

[34]What is Ethereum? `http://www.ethdocs.org/en/latest/introduction/what-is-ethereum.html` (accessed: January 8th 2019)

plans to switch to Proof-of-Stake, where the consensus building process does not rely on solving cryptographic puzzles. In PoS only participants with a legitimate stake can participate, which leads to an increase in performance and scalability[35].

In Ethereum every node gets a copy of the transactions, thereby creating privacy and confidentiality issues. Currently the only option to solve these is by encrypting data locally on the application-level before broadcasting it to the Ethereum network. There are plans to integrate mechanisms, like zk-SNARKS, but it cannot be predicted when these mechanisms will be implemented[36].

Ethereum has the build-in crypto-currency Ether, which is used to pay transaction fees and as reward for miners. Further with Ethereum custom tokens can be created, e.g., to define cryptocurrencies[34].

3.1.2.2 Quorum

Quorum is a permissioned Ethereum implementation, which was developed by J.P. Morgan, focusing on the needs of the financial services industry, but nevertheless, can be used for other industries too. It extends the public Ethereum implementation by following features[37]:

- Transaction and contract privacy[38]: Quorum extends the Ethereum transaction model by the parameter `privateFor` and the transaction type by the new method `IsPrivate`. With the parameter `privateFor` transaction senders can mark who the transaction is private to. The method `isPrivate` can be used to identify private transactions. In contrast to public transactions the payload of private transactions can only be seen by specific network participants. Therefore, the payload is replaced by a hash of the encrypted payload. Authorized participants replace the hash with the original payload by using Constellation instances.
- Multiple voting-based consensus mechanisms: Quorum currently provides two consensus algorithms, Raft-based consensus and Istanbul BFT, which both lead to better scalability.
- Network/Peer permissions management[39]: In Quorum it can be controlled to which nodes a node can connect to and accept connections from. Currently, this whitelisting is done at node-level with a file on the node.

The Quorum architecture consists of the components Quorum Node and Constellation (see Fig. 6). Quorum Node is a fork of the Go-Ethereum (Geth) client with modifications, like replacement of PoW by a vote-based consensus mechanism, node permissioning and support for private transactions. Constellation, consisting of the two modules Transaction Manager and Enclave, is a system for securely submitting information. The Transaction Manager's task is to ensure transaction privacy and confidentiality. Every node's Transaction Manager distributes encrypted transaction

[35] Frequently Asked Questions `http://www.ethdocs.org/en/latest/frequently-asked-questions/frequently-asked-questions.html` (accessed: January 8th 2019)

[36] Ethereum Adoption of zk-SNARK Technology `https://blog.z.cash/ethereum-snarks/` (accessed: January 8th 2019)

[37] Quorum Overview `https://github.com/jpmorganchase/quorum/wiki/Quorum-Overview` (accessed: January 8th 2019)

[38] Transaction Processing `https://github.com/jpmorganchase/quorum/wiki/Transaction-Processing` (accessed: January 8th 2019)

[39] Security `https://github.com/jpmorganchase/quorum/wiki/Security` (accessed: January 8th 2019)

data to authorized participants, stores the encrypted transaction data and manages access to it. Enclave acts as an isolated security module, which is responsible for cryptographic work, e.g., managing of private keys, data encryption, and decryption. With separation of the Transaction Manager and Enclave the performance is improved compared to Ethereum[38].

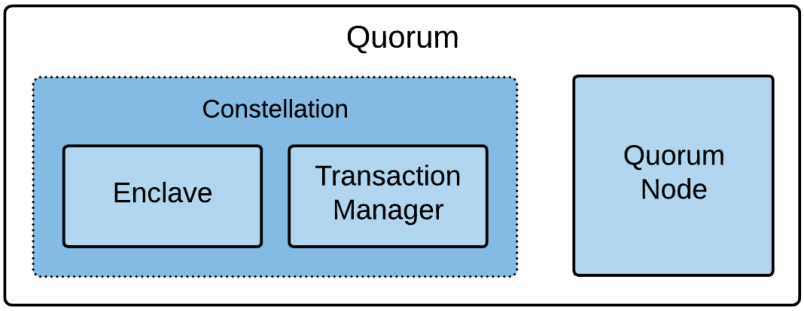

Figure 6: The figure shows the Quorum components. Right is the Quorum Node, a fork of the Go-Ethereum client, and left is the Constellation, consisting of the modules Transaction Manager and Enclave. (based on 37)

Furthermore, Quorum offers a zero-knowledge security layer (ZSL) with help of Zcash, which is based on zk-SNARKS. ZSL-enabled public smart contracts, called z-contracts, can be used to issue digital assets, called z-tokens. These z-tokens are kept private and can be transacted privately[40].

3.1.2.3 Hyperledger Fabric

Hyperledger Fabric is a general-purpose distributed ledger platform implemented by the Linux Foundation. As a permissioned private blockchain it supports a Membership Service Provider (MSP) for managing the network members. Its modular design features pluggable consensus mechanisms, MSPs and encryption mechanisms. Further access control lists (ACL) can be used as an additional permission layer.

Smart contracts in Hyperledger Fabric are realized as chaincodes. Transactions are operations invoked on a chaincode. The chaincode execution is separated from transaction ordering, thereby increasing scalability and performance. Currently chaincodes can be written using the programming languages Go and NodeJS, and are executed by validating nodes inside a Docker container. HLF does not have a built-in cryptocurrency, but currencies and digital tokens can be developed by using chaincodes.

To reach consensus in HLF, two aspects have to be fulfilled, the transaction order and the transaction validity. In HLF, nodes are differentiated by their roles and associated tasks for reaching consensus, however, a single server can run multiple nodes with the different roles, clients, peers, and orderers. While clients act on behalf of the end-user and peers maintain the ledger, orderers are responsible for the transaction order.

[40] ZSL `https://github.com/jpmorganchase/quorum/wiki/ZSL` (accessed: January 8th 2019)

In HLF, privacy and confidentiality can be achieved with channels, where a ledger is only shared between the participants of the channel. This means, that the transactions of a channel are only visible to its participants and not to every participant in the network. Additionally, data can be obfuscated, by encrypting it in a chaincode. A still experimental feature, the Private Data Channel can be used to obscure private data from the network while proving its existence to the network and keeping the data only visible to a specific group of peers[41].

3.1.2.4 Hyperledger Sawtooth

Hyperledger Sawtooth is an open source enterprise blockchain platform under the Hyperledger umbrella. It supports permissioning, where permission settings like roles and identities are stored on the blockchain and every participant can access it. Sawtooth has a highly modular design, thereby enabling application-specific transaction rules, permissioning, and consensus algorithms[42].

In Hyperledger Sawtooth the application level and core system are separated, which eases the development and deployment of applications. Developers can build transaction families, which define transaction types and operations, e.g., smart contracts. Transaction families can be written in a programming language of the developer's choice. Transaction processor software development kits (SDK) are available in multiple programming languages, e.g., Python, JavaScript, Go, C++, Java, and Rust. Further, the Seth project enables interoperability between Sawtooth and Ethereum. It allows the deployment of Ethereum smart contracts to Sawtooth[42].

In Sawtooth, transactions can be processed serially or in parallel, which leads to an improved performance. Sawtooth supports pluggable consensus algorithms, allowing the on-the-fly change of the consensus with a transaction. Currently supported consensus algorithms are Proof-of-Elapsed-Time (PoET), PoET simulator and Dev mode, that are especially optimized for energy-efficiency and scalability[42].

3.1.2.5 Corda

Corda was developed by R3 with the goal to create a global logical ledger for the financial industry. The developers classify Corda not as a blockchain but as a distributed ledger technology. It is a permissioned system, where nodes enforce a peer-to-peer communication [BCGH16].

Smart contracts in Corda can be written in Kotlin, and are validated and run in an augmented Java Virtual Machine (JVM). The smart contract function and data are associated to a legal prose, to make financial agreements legally enforceable. Corda has no built-in cryptocurrency and currently there is no opportunity to define tokens [Hea16].

The transaction data is only shared with associated participants, which results in improved privacy. Therefore, no global consensus is required, only the parties involved in a transaction have to reach consensus over the transaction, which leads to a performance increase compared to other mechanisms. Consensus involves the two aspects of transaction validity and transaction uniqueness. On transaction validity parties agree by running the same contract code and validation logic.

[41] Side DB - Channel Private Data - experimental feature `https://jira.hyperledger.org/browse/FAB-1151` (accessed: January 8th 2019)

[42] Introduction `https://sawtooth.hyperledger.org/docs/core/releases/latest/introduction.html` (accessed: January 8th 2019)

The transaction is valid if the contract code runs successfully, the transaction has all required signatures, and any referring transactions are valid. For transaction uniqueness notary nodes are responsible. Transaction uniqueness means that only the transaction consumes its input states and no other transaction. Therefore, the uniqueness services do not have to see the transaction contents, which improves privacy and scalability. For the uniqueness service, different pluggable consensus algorithms can be used, e.g., BFT, or a single approving machine. In some cases no uniqueness service at all is required[BCGH16][Hea16].

3.1.3 Comparison

As described in Section 3.1.1, as criteria for the selection of a suitable blockchain implementation, following have been chosen:

- Light energy-footprint
- Modular
- Permissioned
- Privacy mechanisms
- Resource-aware
- Smart contracts

While the criteria permissioning, privacy mechanisms, modularity and smart contracts can be evaluated directly, for the light energy-footprint and resource consumption, the consensus mechanism has been considered as indicator. Based on the available information and comparison of the presented blockchain implementations, summarized in Table 1, Hyperledger Fabric seems to present itself as the best choice for the implementation of the SGCBP, because it fulfills the criteria.

The disadvantages of PoW, its openness and no privacy mechanisms disqualify Ethereum as candidate for the platform. While Sawtooth is interesting for IoT devices due to the energy-effcient and highly scalable PoeT consensus mechanism, its lack of privacy mechanisms makes it not suitable for the platform. Corda, with its privacy features, seems like an interesting approach but due to its specific focus on financial use cases is being out of scope. While Quorum as a permissioned fork of Ethereum with implemented PoS and privacy features is interesting, Hyperledger Fabric's advantages prevail.

Hyperledger Fabric as a general-purpose solution is appropriate for the implementation of the platform. The support of a Membership Service Provider and access control lists suits the requirement of permissioning. Further, its modular design and pluggable consensus mechanisms provide the possibility to test different components, such as the consensus mechanism. Including the experimental features, Hyperledger Fabric offers the most possibilities related to privacy. Additional benefits are a good documentation, the community and the use of Docker for running nodes. Using Docker facilitates the development and deployment of the platform. Docker enables to run applications and their dependencies inside containers, which are lightweight and portable across machines. With the opportunity to create tokens, applications involving financial aspects could be implemented into the platform.

On the following pages Hyperledger Fabric and its architecture, by starting with the different nodes, the ledger, and the channels, are described in detail. Then the Membership Service Provider component, chaincodes and the transaction workflow are explained.

Table 1: The table shows an overview on the compared blockchain implementations.

Characteristic / Implementation	Ethereum	Quorum	HL Fabric	HL Sawtooth	R3 Corda
Use Case	Generic	Generic and financial industry	Generic	Generic	Financial industry
Blockchain Type	Permissionless	Permissioned	Permissioned	Permissioned Permissionless	Permissioned
Modularity	no	no	yes	yes	no
Consensus Algorithm	PoW Casper PoS (planned)	Raft-based Istanbul BFT	PBFT SIEVE	PoET PoET simulator Dev mode	Validating nodes Non-validating nodes Distributed Raft
Privacy	No	Private transactions ZSL	Private channels Private data channels	No	Private channels

3.2 Nodes

As already mentioned in Section 3.1.2.3, in Hyperledger Fabric the communication entities of the network are called nodes. These do not represent a physical device, but rather a logical function. This means that on a physical server several nodes can be located. These nodes are classified into the three types clients, peers, and orderers, which are explained in this section.

3.2.1 Peer

Peers are the basic building block of a Hyperledger Fabric network. These are usually owned by organizations and can join multiple channels. Furthermore, peers are responsible to maintain the ledger of each channel. Therefore, each peer keeps a copy of the ledger for each of its channels. When the peer receives blocks from the ordering service with transactions, it commits the changes to its ledger copies. A detailed explanation of channels is given in Section 3.3.2.

Further, peers can be endorsing peers, which host chaincodes. As part of the transaction workflow (described in Section 3.6), these endorsing peers receive transaction proposals from clients and endorse them. This happens by executing chaincode functions according to the proposal and returning the signed chaincode results.

In Hyperledger Fabric, each peer has an identity. Therefore, a digital certificate is required, which is used to identify the peer and its organization. For example, when interacting with a channel, the peer's permissions are retrieved by using the associated identity.

3.2.2 Orderer

The ordering service, consisting of one or multiple orderers, receives transactions, which are then ordered into blocks and sent to the peers to commit to the ledger. The ordering service can be realized in multiple ways, e.g., as centralized orderer or distributed protocol. As of writing, Hyperledger Fabric offers the ordering services SOLO and Kafka, while a third, called Simplified Byzantine Fault Tolerance (SBFT), is planned. The SOLO ordering service is comprised of a single ordering node and mainly used for development. The distributed and fault-tolerant Kafka ordering service, which utilizes Apache Kafka, is recommended to be used for production.

3.2.3 Client

Clients are the applications in the network, which are used by an end-user to connect to the network and read data from or write data to the ledger. Therefore, clients have to communicate with peers. For this purpose, Hyperledger Fabric provides a SDK, which, e.g., enables to create transactions or register for transaction events.

3.3 Hyperledger Fabric Specific Constructs

In this section the two fundamental constructs of HLF, ledgers and channels, are described. A ledger stores information in a blockchain, and channels separate the blockchain network into smaller ones.

3.3.1 Ledger

The ledger in Hyperledger Fabric consists of two related data structures, the state and the chain. The chain is equivalent to the blockchain structure explained in Section 2.4.1. In Hyperledger Fabric it comprises blocks, which contain valid and invalid transactions. Thereby, transactions read and write data on a key-basis. While valid transactions lead to a state change, invalid transactions are stored for audit purposes. Further, a hash over these transactions is part of the block header. Additionally, the block header contains the hash of the previous block header, thereby linking each block to its previous block. The blocks are created by the ordering service, which guarantees that transactions inside a block are totally ordered.

The state represents the recent state of the ledger and is comprised of the recent values for all keys, which are present in the ledger. These key-value pairs are stored in a state database, which is only used by peers. This enables to execute chaincode without the need to process the complete ledger.

Each peer keeps a ledger copy for each of its channels. For state databases two options are implemented, the default LevelDB and the optional CouchDB. In contrast to LevelDB, in CouchDB any binary data can be stored. Further, due to the JSON modelling in CouchDB, data can be stored as JSON and rich queries can be used to query it. These should only be used in transactions, which update data, if it can be guaranteed that the queried data does not change between the chaincode execution and the transaction commit. Furthermore, the CouchDB enables to implement security on field-level by utilizing transaction attributes. Besides LevelDB and CouchDB, there are plans to integrate other databases in Hyperledger Fabric.

3.3.2 Channels

Channels in Hyperledger Fabric separate the blockchain network and allow members to keep their transactions private and confidential. Each channel has its own ledger, which is hosted by the participating peers. To join a channel, the peer has to possess an identity, which is given by a MSP.

Channels are created by using channel configurations, which define all required information, such as the participating organizations, access policies, block size, used hashing algorithm, and orderer addresses. A new channel is created by a genesis block, which contains the initial channel configuration. To change the configuration of a channel, a new configuration block is required. For both cases a configuration transaction (configtx) has to be created. The tool configtxgen can be used to create configtx based on a provided configuration file, the *configtx.yaml*.

If different versions of Hyperledger Fabric exist in a network, transactions would be processed differently. Therefore, capability requirements have been introduced. These define which capabilities are necessary to process transactions and are set for each channel in its channel configuration, e.g., in the configuration file *configtx.yaml*. Three types of capability requirements exist:

- channel capability requirements for peers and orderers,
- orderer capability requirements only for orderers, and
- application capability requirements only for peers.

3.4 Membership Service Provider

Membership Service Providers are responsible for the management of identities. In Hyperledger Fabric these identities are available as X.509 certificates. The MSP defines how these identities can be validated and authenticated. In Hyperledger Fabric two MSP types can be differentiated, local MSPs and channel MSPs. While local MSPs are responsible for clients, peers, and orderers, channel MSPs are used for channels. For each client and node (peer or orderer), a local MSP is used, which is hosted on the client's or node's file system. This local MSP defines administrative and participatory permissions at client or node level, e.g., who has permission to install a chaincode on the peer. In contrast to a local MSP, a channel MSP is stored on each participating node's file system. A channel MSP defines administrative and participatory permissions at channel level, e.g., who has permission to instantiate a chaincode on the channel.

For a MSP several elements are necessary, which can be grouped into following nine directories (depicted in Figure 7):

- Root CAs (RCA): This directory contains self-signed certificates of trusted Root CAs.
- Intermediate CAs (ICA) (optional): This optional directory contains certificates of trusted Intermediate CAs, which have to be signed by a trusted Root CA or Intermediate CA. These Intermediate CAs can be used, e.g., for organization subdivisions.
- Organizational Units (OU) (optional): An optional list of organizational units, which are represented by the MSP, is located in this folder.
- Administrators: In this directory are certificates located, which define the MSP's administrators.
- Revoked Certificates (optional): This optional folder contains a list of certificate revocation lists (CRL). CRLs define which certificates have been revoked and are no further valid.
- Signing Certificates (only for local MSP): This directory contains the node's certificate, which is, e.g., used by an endorsing peer for endorsing a transaction proposal.
- Keystore (only for local MSP): Corresponding to the previous presented directory, it includes the node's signing key, which, e.g., is required for signing transaction proposal responses.
- TLS Root CAs (TLS RCA) (optional): TLS communication requires certificates. This directory contains self-signed certificates of trusted Root CAs used for this purpose.
- TLS Intermediate CAs (TLS ICA) (optional): Similar to the previous directory, located in this directory there are certificates of Intermediate CAs for TLS communication.

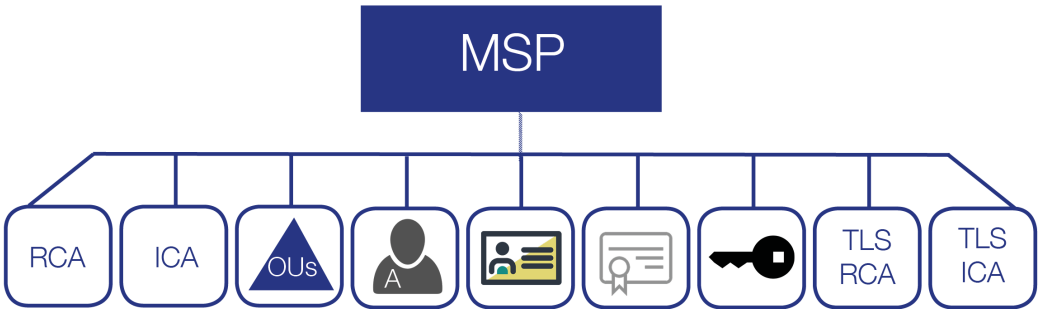

Figure 7: This figure shows the nine MSP directories: Root CAs, Intermediate CAs, Organizational Units, Administrators, Revoked Certificates, Signing Certificates, Keystore, TLS Root CAs, and TLS Intermediate CAs. (based on [Hyp18])

To create the necessary X.509 certificates for the MSP, Hyperledger Fabric provides two options, the cryptogen tool and the Hyperledger Fabric CA. While the cryptogen tool speeds up development and testing, it should only be used for the generation of key material in testing environments. For production, it is advised to use the Hyperledger Fabric CA, which can act as Root CA for the setup of MSPs and enables to dynamically generate and revoke certificates.

3.5 Chaincode

In Hyperledger Fabric smart contracts are realized as chaincodes, which read from or write to the ledger depending on transaction proposals. The created state is thereby only accessible to the specific chaincode. If a chaincode needs access to another chaincode's state on the same channel, the other chaincode needs to be invoked. Further, utilizing chaincodes on other channels is restricted to reading data.

3.5.1 Chaincode Development

As of writing, chaincodes can be written in the programming languages Go and NodeJS. The support of other languages, e.g., Java, is planned. Chaincodes are built into Docker images on endorsing peers and executed in Docker containers.

Each chaincode needs to implement the interfaces `Chaincode` and `ChaincodeStubInterface` from the Hyperledger Fabric shim package[43]. The interface `Chaincode` contains the methods `Init` and `Invoke`, which are called when a transaction is received. The `Init` method instantiates the chaincode's internal data and is executed when an instantiate or upgrade transaction is passed to the chaincode. Further, when an invoke transaction is passed to the chaincode, the method `Invoke` is executed. It requires the name of the chaincode function, which has to be invoked. The interface `ChaincodeStubInterface` contains methods to access the state of the ledger. These methods can be used to read data from the ledger, e.g., with the method `GetState`, and write data to the ledger, e.g., with the method `PutState`.

[43] package shim `https://godoc.org/github.com/hyperledger/fabric/core/chaincode/shim` (accessed: January 8th 2019)

Further, Hyperledger Fabric provides the Blockchain Crypto Service Provider (BCCSP)[44] package, which implements cryptographic standards and algorithms. The offered functions can be grouped into three categories: key lifecycle functions, sign and verify functions, and encrypt and decrypt functions. The implemented cryptographic standards and algorithms are:

- Advanced Encryption Standard (AES)
- Elliptic Curve Digital Signature Algorithm (ECDSA)
- Keyed-Hash Message Authentication Code (HMAC)
- Rivest-Shamir-Adleman (RSA)
- Secure Hash Algorithm (SHA)

When encrypting a value, the invoke transaction has to contain an encryption key in the transient field. Regarding decryption the same key has to be provided, when querying the ledger.

If packages outside the Go standard library, e.g., the BCCSP package, are used, these have to be packaged together with the chaincode. Therefore, the tool govendor can be used, which stores these dependencies in a local directory.

3.5.2 System Chaincodes

A special type of chaincodes are system chaincodes, which are part of the peer. This means, that system chaincodes are deployed together with the peer and do not run in isolated containers. These system chaincodes are following:

- Lifecycle system chaincode (LSCC) for chaincode lifecycle requests
- Configuration system chaincode (CSCC) for peer's channel configuration
- Query system chaincode (QSCC) for querying of ledger data
- Endorsement system chaincode (ESCC) for transaction proposal response signing
- Validation system chaincode (VSCC) for transaction validation

To determine if transactions are correctly endorsed, each peer locally utilizes the VSCC, which checks if received endorsements are valid, their number is sufficient, and their sources are correct. Therefore, endorsement policies are required. These define conditions, based on the identity and role of the signer within the MSP. For example, an endorsement policy could restrict the endorsement of a transaction to specific roles, such as administrators.

[44] package bccsp `https://godoc.org/github.com/hyperledger/fabric/bccsp` (accessed: January 8th 2019)

3.5.3 Chaincode Lifecycle

Before a chaincode can be installed, the chaincode needs to be packaged into a chaincode package. A chaincode package contains the source code of the chaincode, the signatures of the chaincode owners, and optionally an instantiation policy. The default instantiation policy authorizes only admins of the peer's MSP to instantiate the chaincode. Further, if a chaincode has multiple owners, a signed chaincode package (SignedCDS) has to be created and signed by each owner.

To install a chaincode, an install transaction has to be issued by a local MSP administrator of the peer. The transaction puts the source code into a ChaincodeDeploymentSpec and installs it on a specified peer. Thereby, it has to be taken care of the chaincode location, which should be inside the user's *GOPATH*.

To initialize the chaincode, an instantiation transaction needs to be issued. Thereby, it is checked if the transaction creator satisfies the instantiation policy and has write permissions on the channel. During initialization, the chaincode is bound to a specific channel and the endorsement policy setup. After instantiation, transaction proposals can be sent to the chaincode.

Afterwards, the chaincode can be upgraded with upgrade transactions, which set a new chaincode version, which has to be installed on the peer. For all these operations (install, instantiate, and upgrade) the SDK or the CLI on the peers can be used.

3.6 Transaction Flow

A crucial part of HLF is its transaction workflow, which is described in this section. The transaction workflow starts with an application constructing a transaction proposal by using the Hyperledger Fabric SDK. The transaction proposal contains a chaincode invocation request to read data from the ledger, write data to the ledger, or do both, and is signed by the user. This transaction proposal is sent to specific endorsing peers, which are chosen according to the chaincode's endorsement policy. The endorsement policy defines which organizations need to endorse the proposal.

Each endorsing peer inspects the received transaction proposal. To prevent replay-attacks it reviews if the proposal has already been submitted. Further, it checks the signature and if the application is allowed to invoke the chaincode function. If all checks succeeded, each endorsing peer invokes the chaincode according to the transaction proposal. Instead of updating the ledger, the resulting read-write set is signed by the endorsing peer and returned as proposal response to the application. This read-write set consists on one hand of a read set, which contains all read keys and their latest committed values, and on the other hand a write set, which contains all written keys and their new values. Attention has to be paid to the fact, that if the value of a specific key is updated and afterwards read inside the same transaction. Thereby, not the updated value, but rather the last committed value will be read. Further, if a key has to be deleted, a delete marker will be set as value. Nevertheless, the previous committed values for this key will stay in the ledger. To improve scalability, only the endorsing peers have to execute the chaincode during the transaction workflow. To update the ledger, only the returned read and write sets are sufficient.

If the endorsing peers produce different results, e.g., due to different states or a non-deterministic chaincode, the application will receive conflicting transaction proposals. In this case, the application should discard the received proposals and end the transaction workflow. Otherwise, the transaction workflow continues.

In the case of only reading data from the ledger, the application would obtain the queried data from the proposal response and would end the transaction workflow. If data is written to the ledger, the application should check if the endorsement policy has been satisfied. Then it should send a transaction containing the transaction proposal and the proposal responses to the orderer for updating the ledger.

Because multiple applications from several channels can send concurrently transactions to the orderer, the orderer has to order these and bundle them into blocks for each channel. Thereby, the orderer does not inspect the transaction data. This allows the application to encrypt the chaincode input and ouput data or exclude the chaincode input data from the transaction. Further, the transaction order does not need to be equal to the transaction arrival order. In contrast to the general blockchain approach, where ledger forks are possible, this is prevented in Hyperledger Fabric due to the orderer and blocks, which ensure transaction finality.

A created block is then sent to the channel's peers including the endorsing peers. Thereby, it is possible that some peers are not connected to the orderer. Therefore, peers can broadcast the received block to other peers on the channel.

The peers independently handle the block's transactions according to their order in the block. For each transaction the peer checks, if the endorsement policy is satisfied, and if the transaction's read set corresponds to the peer's ledger state to prevent, e.g., double spending threats. More concrete, in case of several transactions updating the same asset, only the first update would be valid and applied. This ensures that each peer has the same ledger. Thereby, the transaction validity is recorded and the block committed to the chain. While valid transactions, in specific their write sets, are applied to the peer's ledger, invalid transactions are only kept for auditing.

If an application is registered for events of a channel, it will receive event notifications for block events and transaction events. For example, an application could register for transaction events, to get notified, e.g., when transactions are committed to the ledger and act upon the received event. A visualization of the complete transaction workflow is depicted in Figure 8.

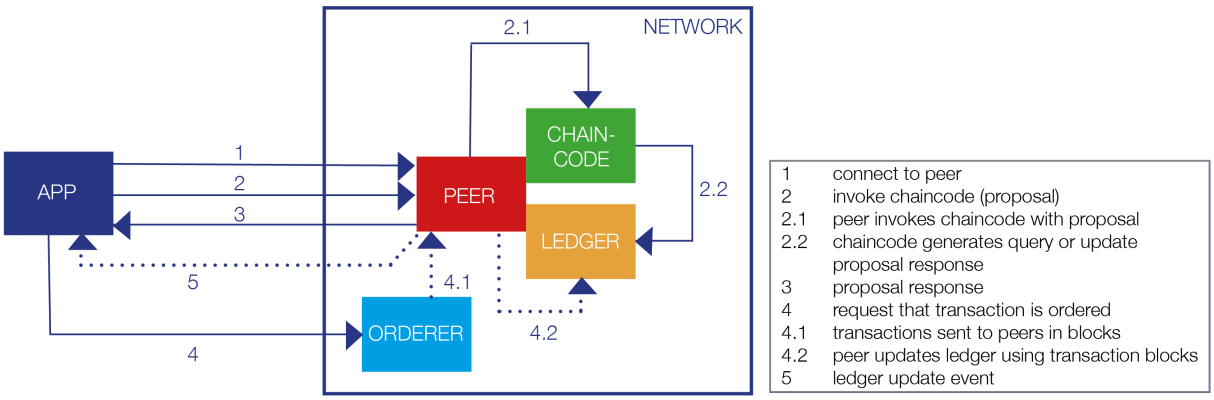

Figure 8: The figure shows the transaction workflow in Hyperledger Fabric from transaction creation to commitment. (based on [Hyp18])

4 Models, Concepts and Hardware

This chapter first introduces the used hardware concepts, and then analyzes a generic system consisting of multiple virtual power plants. The members and processes, relevant to the platform, are being identified and based on this, a model of the platform is developed. This model serves as the basis for the platform implementation. Then the simulation model is described, which includes the simulated environment and data.

4.1 Hardware Concept

This section describes Docker and Kubernetes, which enable to utilize the container technology to create the SGCBP components, such as HLF components and SGCBP applications, and deploy these onto the RPIs. Further, each RPI is connected to a OLED display, which is presented at last.

4.1.1 Docker

Docker is a software written in Go, which enables to create, distribute, and run applications using containers. To run an application in a container, Docker images have to be created, that contain, e.g, the required application code, software libraries, configuration files. These images are then executed within a container[45].

Docker is based on a client-server architecture consisting of the Docker client, the Docker API and the Docker daemon. The Docker daemon is responsible for the Docker objects, like the distribution of containers. The Docker client sends CLI commands to the Docker daemon by using the Docker API. Further, Docker images can be stored in a Docker registry, e.g., the public Docker Hub or an own registry, from which the images can be pulled, which eases the deployment of newer application versions[45]. Due to Hyperledger Fabric utilizing Docker, Docker's advantages, e.g., portability, and compatibility with the Raspberry Pi, Docker is used to run the platform.

[45] Docker overview `https://docs.docker.com/engine/docker-overview/` (accessed: January 8th 2019)

4.1.2 Kubernetes

In 2014 Google open-sourced its Kubernetes project, which is a platform for the management of containers and related workloads and services, such as networking and load-balancing. It automates deploying, scaling, and managing container-based applications on multiple physical or virtual machines. Further, applications can be automatically rolled out and rolled back, monitored and addons used, e.g., DNS and logging mechanisms[46].

Administrators can instruct the master machine with kubectl, which is the Kubernetes command line tool. The master is responsible to control the Kubernetes nodes and assign tasks to them. With pods containers can be grouped and deployed to the nodes. Due to the pod abstracting network and storage from the underlying containers, the pod's containers share network properties, such as IP addresses, and storage. Kubernetes can be used in combination with various container technologies, including Docker. This Kubernetes architecture is shown in Figure 9. Furthermore, services can be used to define access policies for a set of pods[48,28].

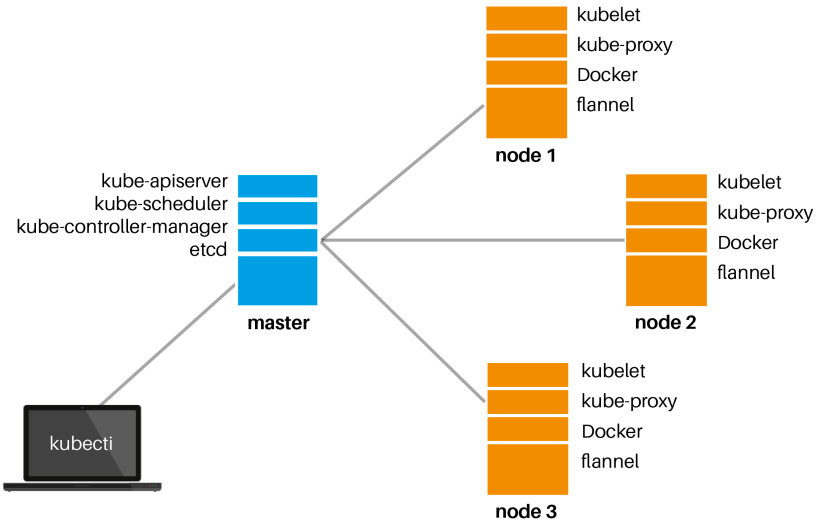

Figure 9: The figure shows a Kubernetes architecture example, consisting of the command line tool kubectl, the master and three nodes. (based on 47)

As Docker is used for the SGCBP components, it can be combined with Kubernetes. Kubernetes enables to roll the SGCBP components out to the RPIs and monitor them. Components, which are realized as containers, can be grouped into a pod and deployed as unit. Furthermore, configuration files can be shared between pods, which eases the setup.

[46] What is Kubernetes? – Kubernetes `https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/` (accessed: January 8th 2019)

[47] Introduction to Kubernetes `https://www.mirantis.com/blog/introduction-to-kubernetes/` (accessed: January 8th 2019)

[48] What is Kubernetes? – Red Hat `https://www.redhat.com/en/topics/containers/what-is-kubernetes` (accessed: January 8th 2019)

4.1.3 OLED Display For Raspberry Pis

To monitor the operation of the SGCBP each RPI will be equipped with an Organic Light Emitting Diode (OLED) display with white pixels and a resolution of 128 x 64 pixels[49]. Raspberry Pi's General Purpose Input/Output (GPIO) pins can be used to interact with external components, e.g., displays, motors and sensors. Some functionalities, e.g., pulse-width modulation (PWM) and inter-integrated circuit (I2C), are only available on dedicated pins.

4.2 System Analysis

The system analysis can be divided into two parts, the Intra-VPP analysis and the Inter-VPP analysis. Whereas, the Intra-VPP analysis focuses only on one VPP and its members and processes, the Inter-VPP analysis considers the processes between multiple VPPs. The mentioned timeslots are fifteen minutes long.

4.2.1 Intra-VPP Analysis

A VPP can include any number of participants with different configurations, e.g., different combinations of devices such as PV systems, battery storages, EVs, and household appliances. These participants, regardless of their configuration, can be classified into producers, consumers, and prosumers. The SGCBP should enable VPP participants to track their energy production and consumption. Therefore, following data has to be stored:

- The timeslot, when the energy was produced or consumed.
- The producing or consuming participant.
- The type, indicating if it is an energy production or consumption.
- The amount of produced or consumed energy in kWh.

Concerning the data granularity, the platform does not focus on processes and devices inside a participant's building. Therefore, the energy production or consumption is aggregated at participant level and only the surplus or deficit is stored on the platform. Further, each VPP has a central instance, the controller, which is responsible for the administration of the platform for the VPP and for aggregating the VPP's energy surplus and demand, based on the reported energy production and consumption data of the participants. Due to the use of personal data on the platform, it has to be ensured that only the participant and their controller have access to the participant's data.

4.2.2 Inter-VPP Analysis

As already mentioned, the controller calculates the VPP's energy surplus or demand and records it on the platform. Therefore, following information is required:

[49] 0.96 inch I2C OLED display - AZ-Delivery `https://www.az-delivery.de/products/0-96zolldisplay` (accessed: January 8th 2019)

- The timeslot of the energy surplus or demand.
- The supplying or demanding VPP.
- The type, indicating if it is an energy surplus or demand.
- The amount of supplied or demanded energy in kWh.

The recorded energy surpluses and demands are then matched together by the platform by creating transactions. This means, that the available energy surpluses of VPPs should satisfy the energy demands of other VPPs, as far as possible. If after the matching an energy supply still exists, the energy is sold to the grid. Otherwise if an energy demand remains, it is satisfied by buying energy from the grid. This matching should be executed for each timeslot and thereby following data should be saved:

- The timeslot of the transaction.
- The selling VPP or the grid.
- The buying VPP or the grid.
- The amount of transacted energy in kWh.

4.3 Platform Model

Based on the system analysis, a model for the SGCBP has been developed. This model defines the underlying Hyperledger Fabric network, the matching algorithm for the energy surpluses and demands, and the used privacy mechanisms for participant data.

4.3.1 Components

To keep the participant's data of a VPP confidential, for each VPP an organization and a channel have to be created. Thereby, members of other VPPs do not have access to it. Further, to each member, in this case a participant or a controller, a peer will be provided. These peers have to join the corresponding channel. Furthermore, per organization a Certificate Authority (CA) is required, which will be hosted by the controller. For the Intra-VPP processes a chaincode has to be developed, which provides the functions to log the participant's energy production and consumption and to query the ledger data by different parameters. To enhance the privacy, the chaincode should enable to encrypt the participant's data in such a way, that only the participant and their controller are able to read the participant's data.

For the Inter-VPP processes, an additional organization and channel have to be setup. This means, that each controller has a second peer, only for the Inter-VPP channel. Regarding the Inter-VPP processes a chaincode is required to enable the logging of VPP energy surplus or demand, the matching of these and the querying of the ledger data. Because chaincode functions need to be invoked, an additional member is necessary to trigger the matching. To this member, the master, a peer is provided, which also joins the Inter-VPP channel. The CA for the Inter-VPP organization and the orderer for the whole HLF network are hosted by the master.

Besides the HLF network, dashboards for the participants, the controllers and the master need to be developed. In their dashboard a participant should see an overview of ones energy production

and consumption. In contrast, the controller's dashboard should show the energy production and consumption of all its participants, and the VPP's energy surpluses, demands, and transactions. The master dashboard should display the logged energy surpluses, demands, and transactions of all VPPs.

4.3.2 Matching

The Inter-VPP chaincode has to implement the functionality to match the logged VPP energy surpluses and demands. Thereby, the highest energy surplus should always be matched to the highest energy demand. Any energy surplus or demand, which remains after the matching, should be matched to the grid. For clarification, the matching algorithm is explained along the following example. In this example, the VPP A provides 5kWh, the VPP C supplies 10 kWh, the VPP B demands 8 kWh and the VPP D requires 7 kWh. This initial situation is summarized in Table 2.

Table 2: The table shows the reported energy surpluses and demands of all VPPs for one timeslot.

VPP	Type	Amount
VPP A	Demand	5 kWh
VPP D	Demand	10 kWh
VPP B	Supply	9 kWh
VPP C	Supply	7 kWh

As described, the highest supply is always matched to the highest demand. In this case, VPP B has the highest supply of 9 kWh, which is matched to the energy demand of 10 kWh of VPP D. Thereby, VPP D's energy demand is reduced to 1 kWh, which can be seen in Table 3.

Table 3: The table shows the remaining energy surpluses and demands after the first matching round, where VPP B's energy supply of 9 kWh was matched to VPP D's energy demand of 10 kWh.

VPP	Type	Amount
VPP A	Demand	5 kWh
VPP D	Demand	1 kWh
VPP C	Supply	7 kWh

After the first matching round, VPP A has the highest energy supply of 5 kWh and VPP C the highest energy demand of 7 kWh. Therefore, they are matched together, whereby an energy supply of 2 kWh for VPP C remains, shown in Table 4.

Table 4: The table shows the remaining energy surpluses and demands after the second matching round, where VPP A's energy demand of 5 kWh was matched to VPP C's energy surplus of 7 kWh.

VPP	Type	Amount
VPP D	Demand	1 kWh
VPP C	Supply	2 kWh

The last remaining energy supply of 2 kWh of VPP C is matched with the last remaining energy demand of 1 kWh of VPP D. The available energy supply of VPP C is thereby reduced to 1 kWh (see Table 5).

Table 5: The table shows the last remaining energy surplus, in this case 1 kWh of VPP C, which is matched to the grid.

VPP	Type	Amount
VPP C	Supply	1 kWh

Finally, the remaining energy supply of 1 kWh of VPP C is sold to the grid. A complete table of all energy transactions is given in Table 6.

Table 6: The table shows all energy transactions between the VPPs and the grid, which have been created as a result of the matching algorithm.

Seller	Buyer	Amount
VPP B	VPP D	9 kWh
VPP C	VPP A	5 kWh
VPP C	VPP D	1 kWh
VPP C	Grid	1 kWh

A workflow diagram of a chaincode function implementing the matching algorithm can be seen in Figure 12.

4.3.3 Privacy Design Strategies

As described in Section 2.3.2, for the implementation of privacy by design multiple privacy design strategies exist. The proof-of-concept puts focuses on data-oriented strategies, which are minimization, hiding, separation, and aggregation. These strategies are applied to personal data, in this case the participant data, and are implemented into the platform as follows:

- Minimization: Only participant's data, which is necessary for the platform operation, should be stored. Therefore, as part of an energylog, only a participant identifier, the date, time, type and amount are stored.
- Hide: To achieve unlinkability and unobservability, encryption and pseudonymization are applied. Pseudonyms are used as participant identifiers and the type and amount of each energy production and consumption is encrypted.
- Seperation: Channels are used for each VPP and an additional channel for the Inter-VPP processes. This enables separation between the VPPs, and the Intra-VPP and Inter-VPP processes. Thereby, the data is kept confidential inside a VPP.
- Aggregation: Only the participant's energy production and consumption, aggregated at participant level, is stored on the platform. More detailed data, such as the energy production or consumption of specific devices, is kept secret by the participant.

4.3.4 Encryption

As described in Section 4.3.3, encryption will be used as one privacy mechanism for the platform. Therefore, symmetric encryption is applied to participant data in the Intra-VPP environment. When an energy production or consumption is recorded, an encryption key has to be passed to

the chaincode. This key is then used by the chaincode to encrypt the type and amount, before writing the data to the ledger. For reading data from the ledger, the key should be passed to the chaincode for decryption.

Further, if the encryption of a text would always return the same result, compare Table 7, this determinism could be used by an attacker to, e.g., execute a known-plaintext attack. In a known-plaintext attack, the plaintext and ciphertext are known to an attacker and they attempt to find out the key by guessing or using an algorithm [TJ11].

Table 7: The table shows the encryption of the text 'hallo' without ciphertext indistinguishability producing always the same output.

Plaintext	Ciphertext
hallo	8labbl0otatspq3xa72k
hallo	8labbl0otatspq3xa72k

Therefore, the encryption method should provide ciphertext indistinguishability. This means, the encryption of a text should never produce the same output, see Table 8. Although, the encryption returns different outputs, they should be decryptable by the same key.

Table 8: The table shows the encryption of the text 'hallo' with cipher indistinguishability producing different outputs.

Plaintext	Ciphertext
hallo	8labbl0otatspq3xa72k
hallo	viumvbprnld4ki9yuqn9

Further, it is advised to use additional encryption on the application-level with an additional encryption key. Thereby, the data should be encrypted before sending it to the chaincode. This approach ensures, that if the encryption key for the chaincode is known to the attacker, the data cannot be read without the encryption key for the application and vice versa. Only the participant and its VPP controller, should know the participant's encryption keys. Furthermore, the controller must have access to the encryption keys of all its participants.

4.4 Simulation Model

The platform is tested by simulating a VPP environment. Simulation applications for the participants, the controllers and the master, are developed. Further, generation and load profiles are required as simulation data for the participants.

Therefore, the standard load profiles utilized by `APCS Power Clearing and Settlement AG (APCS)`[50] will be used. APCS applies these synthetic load profiles for small consumers in forecasting the balancing energy demand of balance groups. These load profiles have been defined by *Verband der Elektrizitätswirtschaft e. V. (VDEW)* and *Energie-Control Austria für die Regulierung der Elektrizitäts- und Erdgaswirtschaft (E-Control)* [EC]. From these profiles, following are used for the simulation:

[50] Synthetische Lastprofile `https://www.apcs.at/de/clearing/technisches-clearing/lastprofile` (accessed: January 8th 2019)

- Load profile for feed-in from hydro, wind and biogas plants (E0)
- Load profile for feed-in from PV plants (E1)
- Load profile for retail stores and hairdressers (G4)
- Load profile for bakeries with a bakehouse (G5)
- Load profile for households (H0)
- Load profile for households with hot water tanks (HA)

Additionally, the load profile from *Mainzer Netze GmbH*[51] for non-switchable electric vehicle charging stations (EL1) is included. All profiles include quarter-hourly values for a year, that are normalized to an annual energy consumption of 1000 kWh.

The simulated environment consists of three VPPs, called Consumers, Mixed, and Producers. Thereby, each VPP includes three participants.

The VPP Producers encompasses only participants with energy sources. A PV farm is operated by participant Alpha, a biogas power plant by participant Bravo and a small wind power plant by participant Charlie. The detailed configuration and used profiles are given in Table 9.

Table 9: The table shows the participants of the VPP Producers, their system configuration and therefore used load profiles.

Participant	Description	Used load profiles
Alpha	3x PV	E1
Bravo	Biogas	E0
Charlie	3x Wind	E0

The VPP Mixed contains an industrial park, consisting of Delta's retail store, Echo's bakery and Foxtrot's PV system. The details of the park and thereby applied load profiles are depicted in Table 10.

Table 10: The table shows the participants of the VPP Mixed, their system configuration and therefore used load profiles.

Participant	Description	Used load profiles
Delta	Retail store	G4
Echo	Bakery with a bakehouse	G5
Foxtrot	PV	E1

The VPP Consumers represents a residential street with three households. While participant Golf only has a general household, Hotel's household is equipped with a hot water tank, and India's household with a non-switchable electric vehicle charging station. This configuration and used load profiles are summarized in Table 11.

[51] Synthetische Lastprofile `https://www.mainzer-netze.de/stromnetze/netzzugang/lastprofile/` (accessed: January 8th 2019)

Table 11: The table shows the participants of the VPP Consumers, their system configuration and therefore used load profiles.

Participant	Description	Used load profiles
Golf	Household	H0
Hotel	Household with hot water tank	HA
India	Household with non-switchable EV charging station	H0, EL1

This configuration was chosen due to the restricted number of available Raspberry Pis and to show the diversity of participants and VPPs. This is done by including consumers and producers as participants, and by grouping, e.g., only consumers or only producers into one VPP. Furthermore, this diversity and normalized profiles enable to thoroughly test the platform operation, especially the matching algorithm.

5 Implementation, Simulation and Results

This chapter describes the implementation of the platform by starting with the adaption of the Hyperledger Fabric components to the Raspberry Pi. Then the developed chaincodes and applications for the platform and simulation are being characterized. This is followed by descriptions of the microSD image for the RPI and the Smart Grid Controller Container Services (SGCCS), which are used to deploy the platform and simulation onto the RPIs. Then it describes the realization of the simulation model, that is described in Section 4.4. Further, the platform operation and stability, and the deployment process were evaluated by using measurements.

5.1 Hyperledger Fabric Components For ARMv7

The RPI is based on the ARM architecture, which HLF does not support out of the box. Further, HLF does not support 32-bit operating systems. This means that the HLF Docker images and binaries have to be created for the RPI. Therefore, the HLF code needs to be modified, which can be found on the following three GitHub repositories:

- hyperledger/fabric-baseimage[52]
- hyperledger/fabric[53]
- hyperledger/fabric-ca[54]

For the thesis, Hyperledger Fabric version 1.1 was used, which corresponds to the tag *1.1.0* on the GitHub repositories.

The initial idea was to use the Raspberry Pi Zero W for the platform. Therefore, HLF had to be ported onto the ARMv6 architecture. Thereby, all attempts have been unsuccessful due to missing dependency support of the ARMv6 architecture. Therefore, it was necessary to opt for a Raspberry Pi model with the ARMv7 architecture, in this case the Raspberry Pi 3 model B,

[52] GitHub repository hyperledger/fabric-baseimage `https://github.com/hyperledger/fabric-baseimage` (accessed: January 8th 2019)

[53] GitHub repository hyperledger/fabric `https://github.com/hyperledger/fabric` (accessed: January 8th 2019)

[54] GitHub repository hyperledger/fabric-ca `https://github.com/hyperledger/fabric-ca` (accessed: January 8th 2019)

where previous HLF versions have been successfully ported onto[55,56]. Furthermore, the Docker Hub frbrkoala[57] contains HLF v1.1 Docker images for the ARMv7 architecture. It was tried to use these images for the SGCBP. But running the basic-network example from the fabric-samples[58] with these images failed due to a runtime error in the fabric-peer container, shown in Code Snippet 1.

```
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x4 pc=0x8a38]

goroutine 107 [running]:
sync/atomic.storeUint64(0x143c9e9c, 0xef46a58c, 0x155e27ee)
    /opt/go/src/sync/atomic/64bit_arm.go:20 +0x3c
github.com/hyperledger/fabric/gossip/identity.(*storedIdentity).fetchIdentity(0
    ↪ x143c9e90, 0x143a11e0, 0x131c0380, 0x20)
    /opt/gopath/src/github.com/hyperledger/fabric/gossip/identity/identity.go
        ↪ :237 +0xa8
```

Code Snippet 1: The code snippet shows the error in the fabric-peer container, when using the Hyperledger Fabric images from the Docker Hub frbrkoala on a 32-bit operating system.

This runtime error is common when using the atomic package[59] on a 32-bit operating system. It can be solved by arranging the 64-bit words for 64-bit alignment. As consequence the fabric code needed to be modified and afterwards the HLF images and binaries built. The build process can be divided, corresponding to the HLF repositories, into three stages. A complete overview of the repositories, their binaries and images is given in Table 12.

[55] Hyperledger Fabric on Raspberry pi 3 `https://stackoverflow.com/questions/45800167/hyperledger-fabric-on-raspberry-pi-3/45804324` (accessed: January 8th 2019)

[56] HYPERLEDGER FABRIC V1.0 ON A RASPBERRY PI DOCKER SWARM - PART 2 `http://www.joemotacek.com/hyperledger-fabric-v1-0-on-a-raspberry-pi-docker-swarm-part-2/` (accessed: January 8th 2019)

[57] frbrkoala's Profile – Docker Hub `https://hub.docker.com/u/frbrkoala/` (accessed: January 8th 2019)

[58] Hyperledger Fabric Samples `github.com/hyperledger/fabric-samples` (accessed: January 8th 2019)

[59] Package atomic `https://golang.org/pkg/sync/atomic/` (accessed: January 8th 2019)

Table 12: The table contains an overview of the Hyperledger Fabric GitHub repositories, the corresponding binaries and Docker images.

HLF GitHub repository	Binaries	Docker images
fabric-baseimage[52]		fabric-baseimage fabric-baseos fabric-basejvm fabric-couchdb fabric-kafka fabric-zookeeper
fabric[53]	gotools peer orderer configtxgen cryptogen configtxlator	fabric-ccenv fabric-javaenv fabric-peer fabric-orderer fabric-tools fabric-buildenv fabric-testenv
fabric-ca[54]	fabric-ca-server fabric-ca-client	fabric-ca fabric-ca-peer fabric-ca-orderer fabric-ca-tools

Regarding the repository fabric-baseimage, the images from the Docker Hub frbrkoala function properly. Therefore, these were used as basis and only the repositories fabric and fabric-ca were modified.

Based on a previous attempt to port HLF onto the RPI[55] changes were made to the fabric repository. The first change concerns the file *\fabric\core\container\util\dockerutil.go* and the discrepancy between the command `uname -m`, used in the images, and `GOARCH`, used in the code, when retrieving the architecture. On the RPI the command `uname` returns the value *arm*, while `GOARCH` returns the value *armv7l*. Therefore, an additional entry in the variable `archRemap` was required, as seen in Code Snippet 2.

Version	Code

old	`var archRemap = map[string]string{`
	`  "amd64": "x86_64",`
	`}`

new	`var archRemap = map[string]string{`
	`  "amd64": "x86_64",`
	`  "arm": "armv7l",`
	`}`

Code Snippet 2: The code snippet shows the change to the file *\fabric\core\container\util\dockerutil.go* due to the discrepancy between the command `uname` and `GOARCH`.

Next, in the configuration file *\fabric\sampleconfig\core.yaml*, which contains runtime settings, the memory limit for Docker containers had to be reduced. In this case, to 16 GB for the RPIs, as seen in Code Snippet 3. The same modification was also applied to the file *\fabric\examples \cluster\config\core.yaml*.

Version	Code
old	`Memory: 2147483648`
new	`Memory: 16777216`

Code Snippet 3: The code snippet shows the changes to the files *\fabric\sampleconfig\core.yaml* and *\fabric\examples\cluster\config\core.yaml* due to the Raspberry Pi's memory limit.

To solve the runtime error, appearing in the frbrkoala images, 64-bit words have to be arranged for 64-bit alignment. In Golang the first field in an allocated struct is automatically 64-bit aligned[59]. Therefore, the position of the field `lastAccessTime` in the struct `storedIdentity` in the file *\fabric\gossip\identity\identity.go* was changed to the first field, as shown in Code Snippet 4.

Version	Code

old	`type storedIdentity struct {`
	`  pkiID common.PKIidType`
	`  lastAccessTime int64`
	`  peerIdentity api.PeerIdentityType`
	`  expirationTimer *time.Timer`
	`}`

new	`type storedIdentity struct {`
	`  lastAccessTime int64`
	`  pkiID common.PKIidType`
	`  peerIdentity api.PeerIdentityType`
	`  expirationTimer *time.Timer`
	`}`

Code Snippet 4: The code snippet shows the first change to the file *\fabric\gossip\identity\identity.go* due to the 64-bit alignment of 64-bit words in Golang.

Due to this change, the function `newStoredIdentity` in the same file, which creates objects of the struct `storedIdentity`, was changed accordingly, which is depicted in Code Snippet 5.

Version	Code
old	`func newStoredIdentity(pkiID common.PKIidType, identity api.PeerIdentityType, expirationTimer *time.Timer) *storedIdentity { return &storedIdentity{ pkiID: pkiID, lastAccessTime: time.Now().UnixNano(), peerIdentity: identity, expirationTimer: expirationTimer, } }`
new	`func newStoredIdentity(pkiID common.PKIidType, identity api.PeerIdentityType, expirationTimer *time.Timer) *storedIdentity { return &storedIdentity{ lastAccessTime: time.Now().UnixNano(), pkiID: pkiID, peerIdentity: identity, expirationTimer: expirationTimer, } }`

Code Snippet 5: The code snippet shows the second change to the file *\fabric\gossip\identity\identity.go* due to the 64-bit alignment of 64-bit words in Golang.

Another necessary change concerns the used Apache Milagro Cryptographic Library (AMCL). When the function `pmul` is called in the file *\fabric\vendor\github.com\milagro-crypto\amcl \version3\go\amcl\FP256BN\FP.go*, an overflow error occurs on the RPI. This happens due to the use of the 64-bit constant `MConst` as `int`, which on the ARMv7 architecture is only 32-bit wide. Therefore, instead of `pmul` the function `pmul64` will be called, which is a modified version of pmul. The function `pmul64` uses the type `int64` instead of the type `int` for the constant, as in Code Snippet 6.

Version	Code
old	`v:=t.pmul(int(MConst))`
new	`v:=t.pmul64(int64(MConst))`

Code Snippet 6: The code snippet shows the change to the file *\fabric\vendor\github.com\milagro-crypto\amcl\version3\go\amcl\FP256BN\FP.go* due to the use of the 64-bit constant `MConst` on a 32-bit operating system.

The function `pmul64` was added to the file *\fabric\vendor\github.com\milagro-crypto\amcl \version3\go\amcl\FP256BN\BIG.go*, as can be seen in Code Snippet 7.

Version	Code

new	`func (r *BIG) pmul64(c int64) Chunk {`
	`  carry:=Chunk(0)`
	`  for i:=0;i<NLEN;i++ {`
	`  ak:=r.w[i]`
	`  r.w[i]=0`
	`  carry,r.w[i]=muladd(ak,Chunk(c),carry,r.w[i])`
	`  }`
	`  return carry`
	`}`

Code Snippet 7: The code snippet shows the change to the file *\fabric\vendor\github.com\milagro-crypto\amcl\version3\go\amcl\FP256BN\BIG.go* due to the use of the 64-bit constant `MConst` on a 32-bit operating system.

To build the Docker images of the fabric repository, the Docker images of the fabric-baseimage repository are required. These are pulled from the Docker Hub Hyperledger Fabric as part of the build process but are not compatible with the RPI. Therefore, the Docker images from the Docker Hub frbrkoala should be used, which requires to disable the pulling from the Docker Hub Hyperledger Fabric. This is done by changing the *Makefile*, as shown in Code Snippet 8.

Version	Code
old	`docker: docker-thirdparty $(patsubst %,build/image/%/$(` `  ↪ DUMMY), $(IMAGES))`
new	`docker: $(patsubst %,build/image/%/$(DUMMY), $(IMAGES))`

Code Snippet 8: The code snippet shows the change to the *Makefile* to disable the pulling of Docker images from the Docker Hub Hyperledger Fabric, which are not compatible with the Raspberry Pi.

After the changes were completed, the fabric binaries and images had to be created, by running the commands `make native` and `make docker`. Thereby, the build process had to be restarted several times, due to a lack of memory on the RPI.

As next, the binaries and images of the fabric-ca repository, which comprises the HLF certificate authority components, had to be created. During the build process of the images, although it should, the version of the fabric-ca-client and the fabric-ca-server was not passed to the file *\fabric-ca\lib\metadata\version.go*. As workaround, the version was hardcoded into this file, as in Code Snippet 9.

Version	Code
old	`var Version`
new	`var Version string = "1.1.0"`

Code Snippet 9: The code snippet shows the change to the file *\fabric-ca\lib\metadata\version.go* due to the missing version of the fabric-ca-client and fabric-ca-server.

The fabric-ca-server uses the libltdl library, which can be found in the libtool package[60]. Therefore, the libtool package has to be installed when building the fabric-ca-server image. For this purpose, the argument `FABRIC_CA_DYNAMIC_LINK` has to be set to true when the fabric-ca binaries and images are created by running the commands `make fabric-ca-server`, `make fabric-ca-client` and `make docker`. Similar to the build process for the fabric repository, steps of the build process needed to be repeated several times due to memory limits of the RPI.

The created images were tested by running the basic-network example and the first-network example from the fabric-samples repository. All HLF images were pushed to the Docker Hub goranovic[61]. Furthermore, the modified GitHub repositories are available under the following links:

- goranovic/fabric[62]
- goranovic/fabric-ca[63]

After the HLF binaries and images, the chaincodes and applications had to be developed. After considering Hyperledger Composer[64], a development toolset and framework for the development of HLF blockchain applications, due to the missing support of multi-channel access, which is required for the creation of the platform, instead of using the Hyperledger Composer the chaincode and applications were developed manually[65].

5.2 Chaincodes

As explained in Section 4.3.1, two chaincodes are required for the platform. The Intra-VPP chaincode sgcbpintra is responsible for processes inside a VPP and the Inter-VPP chaincode sgcbpinter for processes between VPPs. Chaincodes in HLF can be written in NodeJS or Golang. Due to experiences with NodeJS, the chaincodes were first written in NodeJS. But executing the chaincodes on the RPI failed with a timeout error, shown in Code Snippet 10.

```
[chaincode] launchAndWaitForRegister -> DEBU 54f[0m stopping due to error while
    ↪ launching: timeout expired while starting chaincode
```

Code Snippet 10: The code snippet shows the appearing timeout error when using chaincode written in NodeJS.

The source of this problem could not be identified. Therefore, the chaincodes were ported to Golang, where the error did not appear again.

[60] GNU Libtool - GNU Project - Free Software Foundation `https://www.gnu.org/software/libtool/` (accessed: January 8th 2019)
[61] goranovic's Profile – Docker Hub `https://hub.docker.com/u/goranovic` (accessed: January 8th 2019)
[62] goranovic/fabric `https://github.com/goranovic/fabric` (accessed: January 8th 2019)
[63] goranovic/fabric-ca `https://github.com/goranovic/fabric-ca` (accessed: January 8th 2019)
[64] Hyperledger Composer - Create business networks and blockchain applications quickly for Hyperledger — Hyperledger Composer `https://hyperledger.github.io/composer/latest/` (accessed: January 8th 2019)
[65] Multi-Channel Support `https://github.com/hyperledger/composer/issues/2103` (accessed: January 8th 2019)

As described in Section 3.5.1, each chaincode has to implement the interface `Chaincode` and its methods `Init` and `Invoke`. The method `Init` is called when a chaincode is instantiated or upgraded and initializes the chaincode on a channel. The method `Invoke` represents the entry point of the chaincode and is called each time a chaincode is invoked by an invoke transaction, e.g., when updating or querying the ledger. Based on the transaction, which includes the identifier of the targeted chaincode method, the `Invoke` method calls the corresponding method and passes the arguments from the transaction to it.

Furthermore, each chaincode has to implement the interface `ChaincodeStubInterface`, which contains multiple methods to read from and write to the ledger. The developed chaincodes sgcbpintra and sgcbpinter only utilize the methods `GetState`, `PutState`, and `GetQueryResult`.

Data is written as key-value pair with the method `PutState`. The method puts the data as a data-write proposal into the writeset of the transaction. This means, as long as the transaction is not committed, the data will not be written to the ledger. To read from the ledger, the method `GetState` is used, which returns the value corresponding to the passed key.

Due to the use of a CouchDB state database, rich queries can be performed [Hyp18]. Therefore, all query methods in the chaincodes use the method `GetQueryResultForQueryString`, which takes a CouchDB query string as argument. An example for a query string for querying energylogs by a timeslot, is provided in Code Snippet 11.

```
{"selector":{"docType":"energylog","date":<DATE>,"time":<TIME>"}
```

Code Snippet 11: The code snippet shows a CouchDB query string for querying energylogs by date and time.

The method `GetQueryResultForQueryString` passes the query string to the method `GetQueryResult` of the interface `ChaincodeStubeInterface`, which performs a rich query against the state database and returns an interator over the query result set. The query result set is then iterated and a JSON array created, which contains the query results. Finally, this array is returned as a byte buffer.

Because chaincodes are executed in Docker containers, Docker images for the chaincodes are necessary. These are built the first time when the chaincode is executed on a peer. If a chaincode execution is triggered, e.g., a chaincode function needs to be invoked, while the chaincode image is built and the building takes too long, the execution will be aborted. This situation is called premature execution. To prevent it, each chaincode requires an initialization method, that is only called to trigger the building of the chaincode image. This initialization method should be called as part of the deployment, before the applications start to operate[66].

5.2.1 Chaincode sgcbpintra

The Intra-VPP-chaincode sgcbpintra focuses on the participant's energy production and consumption. As described in Section 4.2.1, following information has to be stored: participant, timeslot, type, and amount. Therefore, the struct `Energylog`, depicted in Code Snippet 12, is created.

[66] Chaincode image (accessed: January 8th 2019)

```
type Energylog struct {
       ObjectType string `json:"docType"`
       Participant string `json:"participant"`
       Date string `json:"date"`
       Time string `json:"time"`
       Etype string `json:"etype"`
       Amount string `json:"amount"`
}
```

Code Snippet 12: The code snippet shows the struct `Energylog` from the chaincode sgcbpintra.

To create an energylog the method `createEnergylog` has to be called. It creates an energylog with the key format `PARTICIPANT_DATE_TIME`. For querying existing energylogs multiple methods exist, as can be seen in the sgcbpintra class diagram from Figure 10.

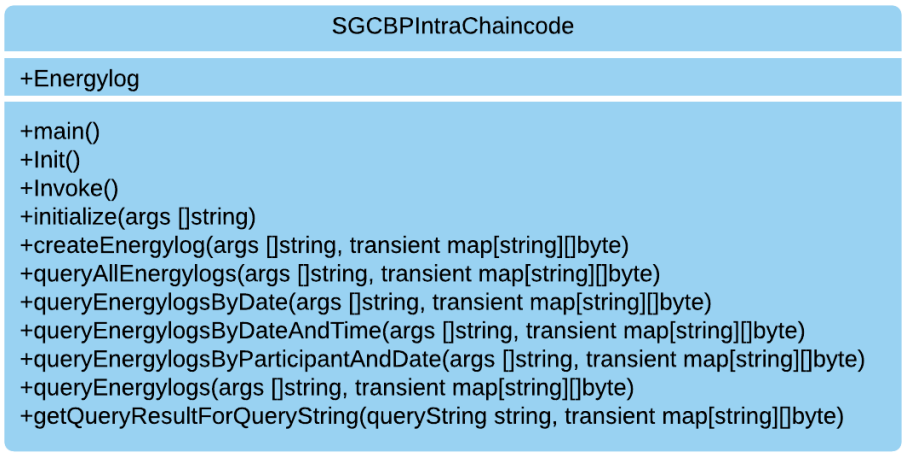

Figure 10: The figure shows the class diagram for the sgcbpintra chaincode.

In contrast to the sgcbpinter chaincode, the sgcbpintra chaincode integrates encryption and decryption for the amount and the type in the struct `Energylog`. Therefore, following Hyperledger packages are utilized:

- Blockchain Crypto Service Provider (BCCSP) (see Section 3.5.1)
- BCCSPFactory[67]: creates instances of BCCSP
- Entities[68]: contains methods for cryptographic operations

When creating an energylog, an encryption key has to be passed as transient field. As encryption algorithm, AES-256 is used. Therefore, an `AES256EncrypterEntity` object is created with the encryption key as argument. While the encryption key is a mandatory argument, an initialization

[67] package factory `https://godoc.org/github.com/hyperledger/fabric/bccsp/factory` (accessed: January 8th 2019)

[68] package entities `https://godoc.org/github.com/hyperledger/fabric/core/chaincode/shim/ext/entities` (accessed: January 8th 2019)

vector can be passed optionally. If an IV is not provided, the `AES256EncrypterEntity` object generates the IV randomly. In an environment, where multiple endorsing peers receive a chaincode invoke request for the creation of an energylog and the IV is generated randomly by the chaincode, the creation would fail. Each endorsing peer would generate a different IV and thereby encrypt the plaintext differently. Because transaction proposals are only sent to one endorsing peer in the case of the SGCBP, this limitation does not apply. Therefore, only the encryption key is passed and the IV is generated randomly by the chaincode, which is then stored together with the ciphertext. When querying the ledger, the same encryption key has to be used for decrypting the ciphertexts. Furthermore, when querying energylogs of multiple participants, a mapping of the participants and their keys has to be provided.

5.2.2 Chaincode sgcbpinter

The Inter-VPP chaincode sgcbpinter handles the energy surplus or demand of the VPPs and the transactions between them. As characterized in Section 4.2.2, for the energy production and consumption, VPP, timeslot, type and amount have to be recorded. This is realized by using the struct `Vpplog`, which is shown in Code Snippet 13.

```
type Vpplog struct {
	ObjectType string `json:"docType"`
	Vpp string `json:"vpp"`
	Date string `json:"date"`
	Time string `json:"time"`
	Etype string `json:"etype"`
	Amount int `json:"amount"`
}
```

Code Snippet 13: The code snippet shows the struct `Vpplog` from the chaincode sgcbpinter.

For the transactions, which require buyer, seller, timeslot, and amount, the struct `Vpptransaction` has been created, as depicted in Code Snippet 14.

```
type Vpptransaction struct {
	ObjectType string `json:"docType"`
	Seller string `json:"seller"`
	Buyer string `json:"buyer"`
	Date string `json:"date"`
	Time string `json:"time"`
	Amount int `json:"amount"`
}
```

Code Snippet 14: The code snippet shows the struct `Vpptransaction` from the chaincode sgcbpinter.

For the creation of a vpplog the method `createVpplog` has to be called, and for the generation of vpptransactions the method `createSettlement` is called by the master. For vpplogs the key format is `VPP_DATE_TIME` and for vpptransactions `SELLER_BUYER_DATE_TIME`. The method

`createSettlement` executes the matching of energy surpluses and demands of the VPPs for a given timeslot. It checks if a settlement already exists for the specified timeslot. If no settlement exists, all vpplogs for the specified timeslot are retrieved. These vpplogs are then split into energy surpluses and demands. Then the matching algorithm, as described in Section 4.3.2, is executed. Finally, the generated vpptransactions are written to the ledger. The workflow of `createSettlement` is visualized as a flow diagram in Figure 12.

Furthermore, the vpplogs and vpptransactions can be queried with multiple methods, which are listed in the sgcbpinter class diagram in Figure 11.

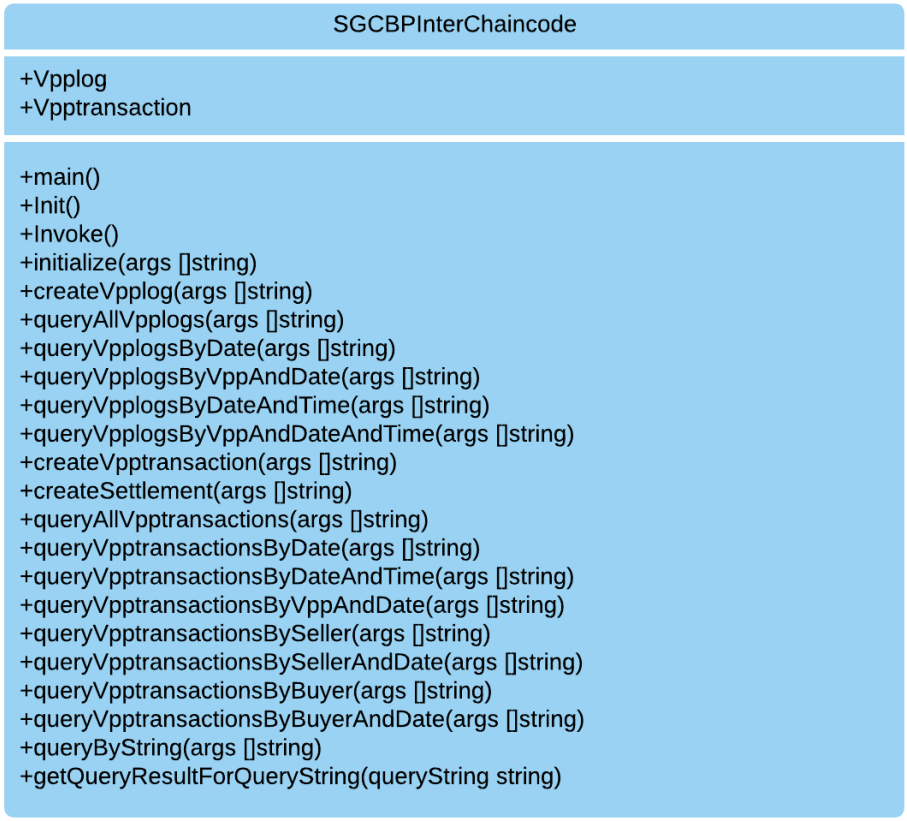

Figure 11: The figure shows the class diagram for the sgcbpinter chaincode.

The source code of both chaincodes is available on the GitHub repository goranovic/sgcbp[69].

[69] GitHub repository goranovic/sgcbp `https://github.com/goranovic/sgcbp` (accessed: January 8th 2019)

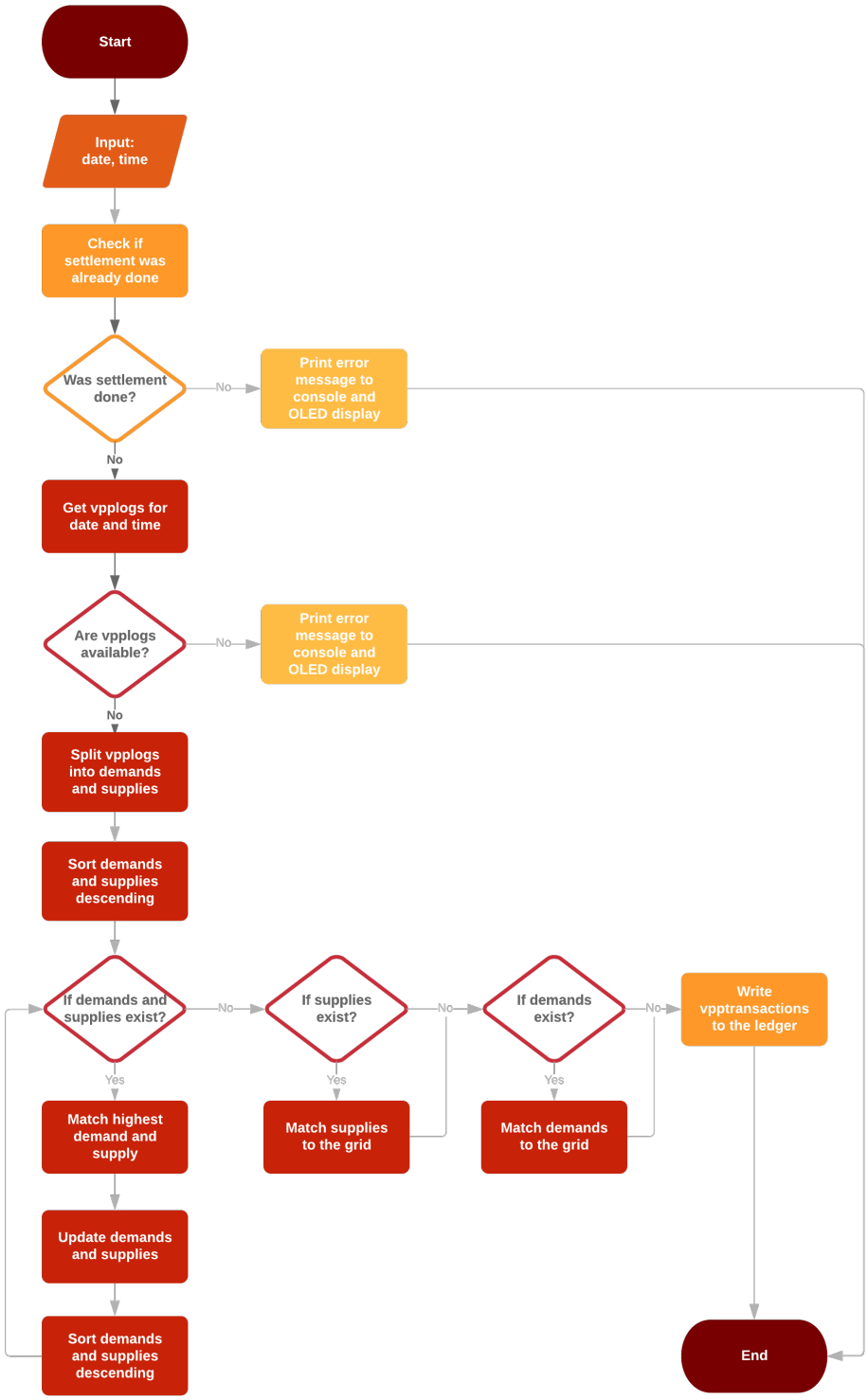

Figure 12: The figure shows the workflow of the function `createSettlement`, which is used to match VPP's energy surpluses and demands.

5.3 Applications

The applications for the platform and the simulation, which operate on top of the HLF network, can be divided into following three categories:

- Application Programming Interface (API) server: acts as connector to the HLF network by providing endpoints for the chaincode functions.
- Simulation: simulates the functionalities of the participant, the controller or the master.
- Dashboard: displays different data depending on the role in the platform, e.g., participant's energy consumption and production.

All applications were written in NodeJS and built into Docker images. On the following pages, the developed applications are described in detail.

5.3.1 API

To provide access to the HLF network, API applications for the sgcbpintra chaincode (sgcbpintra-API) and the sgcbpinter chaincode (sgcbpinter-API) were developed. For the development of the API applications an online article[70] was used as starting point. These applications offer endpoints to the functions of the chaincodes. For the development, Express[71], a web framework for NodeJS, was used, which eases the development of APIs by providing HTTP utility methods. Further, to parse JSON request bodies into objects the Node module body-parser[72] is utilized. These objects are passed as arguments to functions of the class `SGCBPNetwork`. These functions create transaction proposals based on the passed objects by adding necessary data, such as the chaincode id, the transaction id and the chaincode function name.

To submit a transaction or to query the ledger, the transaction is passed to the class `FBClient`, which extends the Node module fabric-client[15]. When submitting a transaction, first a transaction proposal is sent to an endorsing peer. Then the endorsing peer executes the chaincode according to the received transaction proposal and returns a proposal response, indicating if the proposal should be endorsed. If it should be endorsed, the proposal and proposal response are sent to the orderer. By using a `ChannelEventHub` object, a transaction listener for the transaction is registered. Thereby, the application will be notified, when the transaction is finally commited to the ledger. When querying the ledger, a transaction proposal is sent to an endorsing peer. The endorsing peer invokes then the required chaincode function and returns the result.

Connection profiles are required for the connection to the HLF network. A connection profile contains a description of the HLF network, which the client wants to access[73]. Furthermore, the connection profile contains the path to the user certificates, which are used for interacting with the HLF network. The users have to be registered and the user certificates have to be created in advance.

[70] Setting up RESTful API Server for Hyperledger Fabric With NodeJS SDK `https://hackernoon.com/setting-up-restful-api-server-for-hyperledger-fabric-with-nodejs-sdk-a4642edaf98e` (accessed: January 8th 2019)

[71] Express - Node.js web application framework `https://expressjs.com/` (accessed: January 8th 2019)

[72] body-parser - npm `https://www.npmjs.com/package/body-parser` (accessed: January 8th 2019)

[73] How to use a common connection profile `https://fabric-sdk-node.github.io/tutorial-network-config.html` (accessed: January 8th 2019)

Besides the provided endpoints, the sgcbpintra-API and sgcbpinter-API differ in their requests. For example, the sgcbpintra-API requires encryption and decryption keys, which are passed to the sgcbpintra chaincode as transient field, whereas the sgcbpinter-API does not. A list of all API endpoints and their request type is given in Table 13.

Table 13: The table lists all endpoints for the chaincodes sgcbpintra and sgcbpinter, which are provided by the API applications.

sgcbpintra API endpoints	sgcbpinter API endpoints
POST createEnergylog	POST createVpplog
POST queryAllEnergylogs	GET queryAllVpplogs
POST queryEnergylogsByDate	POST queryVpplogsByDate
POST queryEnergylogsByDateAndTime	POST queryVpplogsByVppAndDate
POST queryEnergylogsByParticipantAndDate	POST queryVpplogsByDateAndTime
POST queryEnergylogs	POST queryVpplogsByVppAndDateAndTime
	POST createVpptransaction
	POST createSettlement
	GET queryAllVpptransactions
	POST queryVpptransactionsByDate
	POST queryVpptransactionsByDateAndTime
	POST queryVpptransactionsByVppAndDate
	POST queryVpptransactionsBySeller
	POST queryVpptransactionsBySellerAndDate
	POST queryVpptransactionsByBuyer
	POST queryVpptransactionsByBuyerAndDate
	POST queryByString

5.3.2 Simulation

The simulation applications simulate the functionality of a participant, a controller or a master. The basic workflow of each simulation application is the same and is executed at a set interval. Therefore, the Node module Cron[74] is used, which allows to schedule tasks to run at a specified interval.

Each interval, the application retrieves the current time by using the Node module moment[75]. The retrieved timestamp in the format *MM:ss*, e.g., 00:15, is used as index in the corresponding dataset. As a consequence of this specification, the maximum simulation duration is one hour. The dataset should be provided as a CSV file. For reading and querying the CSV file by the index, the Node module node-csv-query[76] is utilized.

Depending on the simulated behavior, the simulation application sends requests to the API applications, e.g., to create an energylog. For the requests to the API applications the HTTP client Axios[77] is used. Furthermore, the Node module rpi-oled[78] enables to display messages on

[74] cron - npm `https://www.npmjs.com/package/cron` (accessed: January 8th 2019)
[75] Moment.js — Home `http://momentjs.com/` (accessed: January 8th 2019)
[76] node-csv-query - npm `https://www.npmjs.com/package/node-csv-query` (accessed: January 8th 2019)
[77] axios - npm `https://www.npmjs.com/package/axios` (accessed: January 8th 2019)
[78] NodeJS library and command line tools for controlling SSD1306 compatible I2C OLED screens on the Raspbery Pi `https://github.com/normen/rpi-oled` (accessed: January 8th 2019)

the connected OLED display. An example is shown in Figure 13, where the participant simulation displays energylog data on the OLED display.

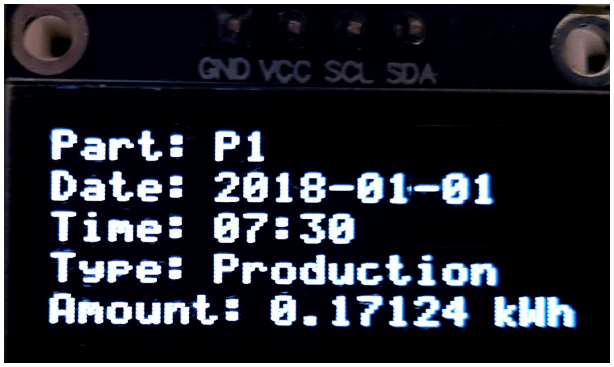

Figure 13: The figure shows a picture of a participant's OLED display, which is displaying information about an energylog.

5.3.2.1 Participant-Simulation

The participant simulation simulates a participant, who periodically logs their energy production or consumption. Therefore, the provided dataset consists of a table of energylog data. An example of such a dataset can be seen in Table 14.

Table 14: The table shows a sample of a participant dataset.

index	date	time	type	amount
00:00	2018-01-01	00:00:00	Consumption	4
00:15	2018-01-01	00:15:00	Consumption	5
00:30	2018-01-01	00:30:00	Production	7
00:45	2018-01-01	00:45:00	Consumption	3

Before sending a POST request to the sgcbpintra-API with the energylog data, the simulation encrypts the type and amount of the energylog. As in the sgcbpintra-CC, the simulation uses AES-256 as encryption algorithm. While the encryption key, called APP-encryption-key, is provided, the IVs are randomly generated. For the encryption and decryption, the Node module crypto[79] is used. Thereby, each ciphertext and its IV are stored together in the format *IV:ciphertext*.

The application then sends a POST request to the sgcbpintra-API endpoint `createEnergylog` with the energylog data as argument in the request body. Thereby, another encryption key, the CC-encryption-key, has to be passed as transient field to the chaincode for chaincode encryption.

If the data has been written successfully to the ledger, the energylog details will be displayed on the console and the display. Otherwise, an error message will appear. A flow diagram of the participant simulation workflow can be seen in Figure 14.

[79] Crypto — Node.js v11.6.0 Documentation `https://nodejs.org/api/crypto.html` (accessed: January 8th 2019)

5.3.2.2 Controller-Simulation

The controller's task is to calculate its VPP's energy surplus or demand and log it on the ledger. Therefore, the controller operates after the participants have logged their energy consumption and production. The controller only requires the date and time for its operation, as the example of a controller dataset in Table 15 shows.

Table 15: The table shows a sample of the controller dataset.

index	date	time
00:07	2018-01-01	00:00:00
00:22	2018-01-01	00:15:00
00:37	2018-01-01	00:30:00
00:52	2018-01-01	00:45:00

After reading a dataset record, the application sends a POST request to the sgcbpintra-API endpoint `queryEnergylogsByDateAndTime` to retrieve energylogs for the specified timeslot. Due to encryption of the energylogs, the POST request has to include a mapping of all participants and their CC-encryption-keys.

Furthermore, the retrieved energylogs have to be additionally decrypted by the simulation. Thereby, the same Node module crypto, as for the participant simulation, is used. As with the chaincode decryption, the controller requires for the decryption, a mapping of participants and their APP-encryption-keys. Depending on the field `participant` of an energylog, it finds the corresponding APP-encryption key and decrypts the fields `type` and `amount`.

The application then calculates the VPP's energy surplus or demand and creates a vpplog by sending a POST request with the vpplog data to the sgcbpinter-API endpoint `createVpplog`. The details of the vpplog will be printed on the console and the display, if the vpplog has been written successfully to the ledger, otherwise an error message will be displayed. The workflow of the controller simulation can be seen in Figure 14.

5.3.2.3 Master-Simulation

The master is only responsible to trigger the matching, which creates vpptransactions. Comparable to the controller dataset, the master dataset contains date and time values. The master triggers the matching after controllers have logged their VPP's energy surpluses and demands. Therefore, compared to the participant dataset, the master dataset is shifted by one timeslot, as depicted in Table 16.

Table 16: The table shows a sample of the master dataset.

index	date	time
00:15	2018-01-01	00:00:00
00:30	2018-01-01	00:15:00
00:45	2018-01-01	00:30:00
01:00	2018-01-01	00:45:00

To trigger the matching, the application sends a POST request to the sgcbpinter-API endpoint `createSettlement` with date and time provided in the request body. If the matching has been

successful, the application queries the created vpptransactions by the same date and time. Therefore, a POST request is sent to the sgcbpinter-API endpoint `queryVpptransactionsByDateAndTime` with the date and time in the request body. If vpptransactions exist and no error occurred, the queried vpptransactions will be displayed on the console and display, otherwise an error message is shown. The master simulation workflow is summarized as a flow chart in Figure 14.

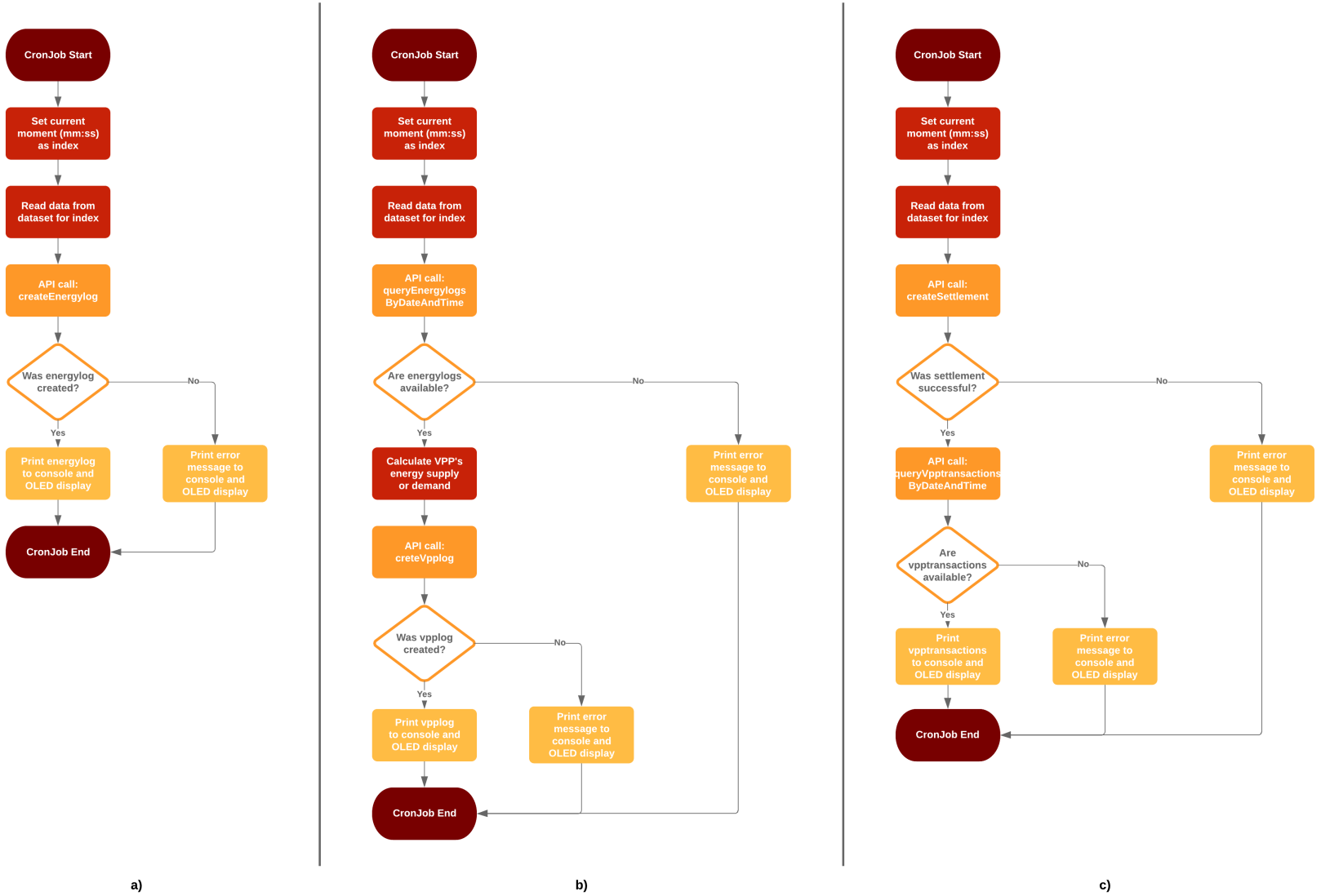

Figure 14: The figure shows the workflows of the simulations, starting with the (a) participant simulation workflow, then the (b) controller simulation workflow is depicted and finally the (c) master simulation workflow.

5.3.3 Dashboard

The dashboard displays data from the ledger to a participant, a controller or the master. For the development of the dashboards ReactJS, a library for building UI, was used. Further, for the development of the dashboard applications an online article[80] was used as starting point. The dashboard is composed of different components, which can be individually adopted. For the different dashboards following components were developed:

- Calendar component: With the calendar component the user can set the date, which is used as parameter for querying data from the ledger. Therefore, it utilizes the reactdatepicker[81] component. Further, the dashboard can operate in two modes, the LIVE mode and the DATE mode. While in the LIVE mode the dashboard is periodically refreshed and data requeried, this refreshing is disabled in the DATE mode. To be able to switch the mode a button was added.
- Table component: The table component is used to display data in a table, e.g., energylog data of participants. The data can be sorted ascending or descending by the columns. For the implementation the bootstrap-table[82] component was used.
- Logo component: The SGCBP logo is displayed by using the logo component.
- Bar component: To display data over a period as a bar chart, like participant's energy production and consumption, the bar component can be used. It utilizes the React wrapper for Chart.js 2[83]. When the user hovers over a bar, a box with corresponding information appears.
- Pie component: With the pie component, data can be displayed as a pie chart, e.g., the VPP's overall energy production per participant. Comparable to the bar component, the pie component uses the React wrapper for Chart.js 2. When hovering over a segment of the pie, further details are shown in a box.

The calendar and logo component are present in every dashboard. In the following, the dashboards for the different roles and the displayed data are presented.

5.3.3.1 Participant-Dashboard

The participant dashboard displays information about the participant's logged energy production and consumption, as can be seen in Figure 15. The dashboard is composed of following components:

- Table component shows the energylog data of the set date.
- Bar component displays the energy production and consumption over the specified date.

[80] How to create a dashboard app with React — Creative Bloq `https://www.creativebloq.com/how-to/create-a-dashboard-app-with-react` (accessed: January 8th 2019)

[81] ReactJS Datepicker crafted by HackerOn `https://reactdatepicker.com/` (accessed: January 8th 2019)

[82] Components – React-Bootstrap `https://react-bootstrap.github.io/components/table/` (accessed: January 8th 2019)

[83] React wrapper for Chart.js `https://github.com/jerairrest/react-chartjs-2` (accessed: January 8th 2019)

Regarding the future development, two pie components were added for energy production and consumption data on device-level. While the first pie component shows the amount of energy production per device, the other pie component illustrates the amount of energy consumption per device. For the proof-of-concept, the pie components display only dummy data.

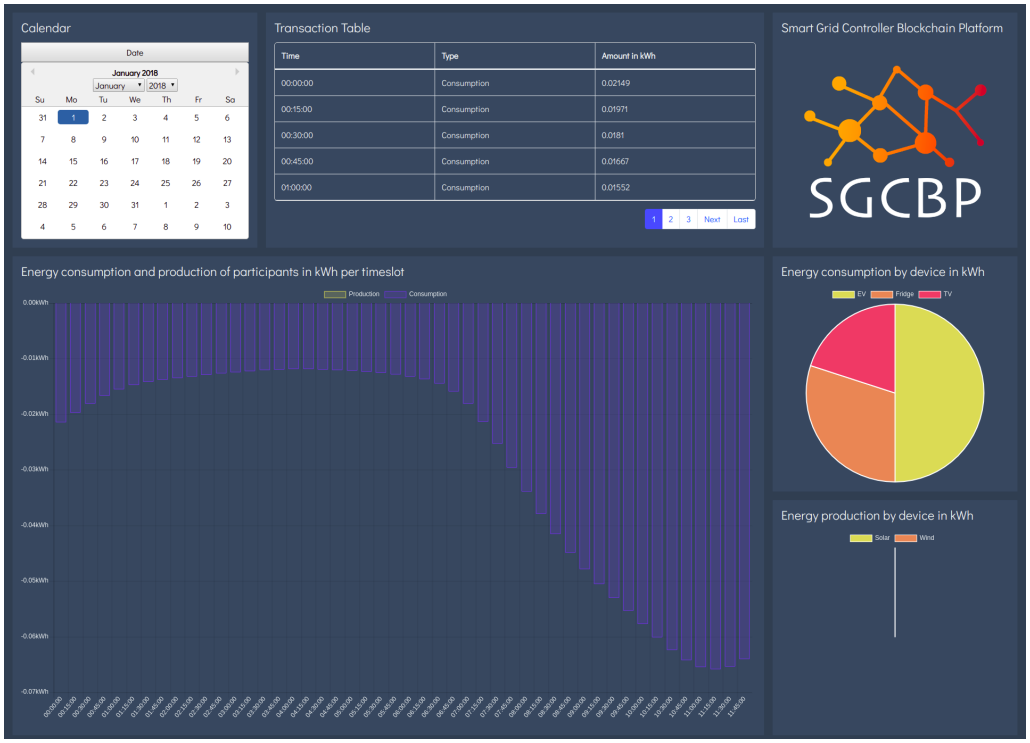

Figure 15: The figure shows the participant dashboard, which displays information about the participant's energy production and consumption.

5.3.3.2 Controller-Dashboard

The controller dashboard shows information regarding its VPP participants, their logged energy production and consumption and the VPP's transactions, as shown in Figure 16. It consists of following components:

- Two table components:
  - the first portrays the energylogs of the VPP participants, and
  - the other displays the VPP's transactions.
- Bar component represents the VPP participant's energylog data.
- Four pie components:
  - the first illustrates the amount of the VPP's energy production per participant,
  - the second shows the amount of VPP's energy consumption per participant,
  - the third represents the amount of bought energy per seller, and
  - the last visualizes the amount of sold energy per buyer.

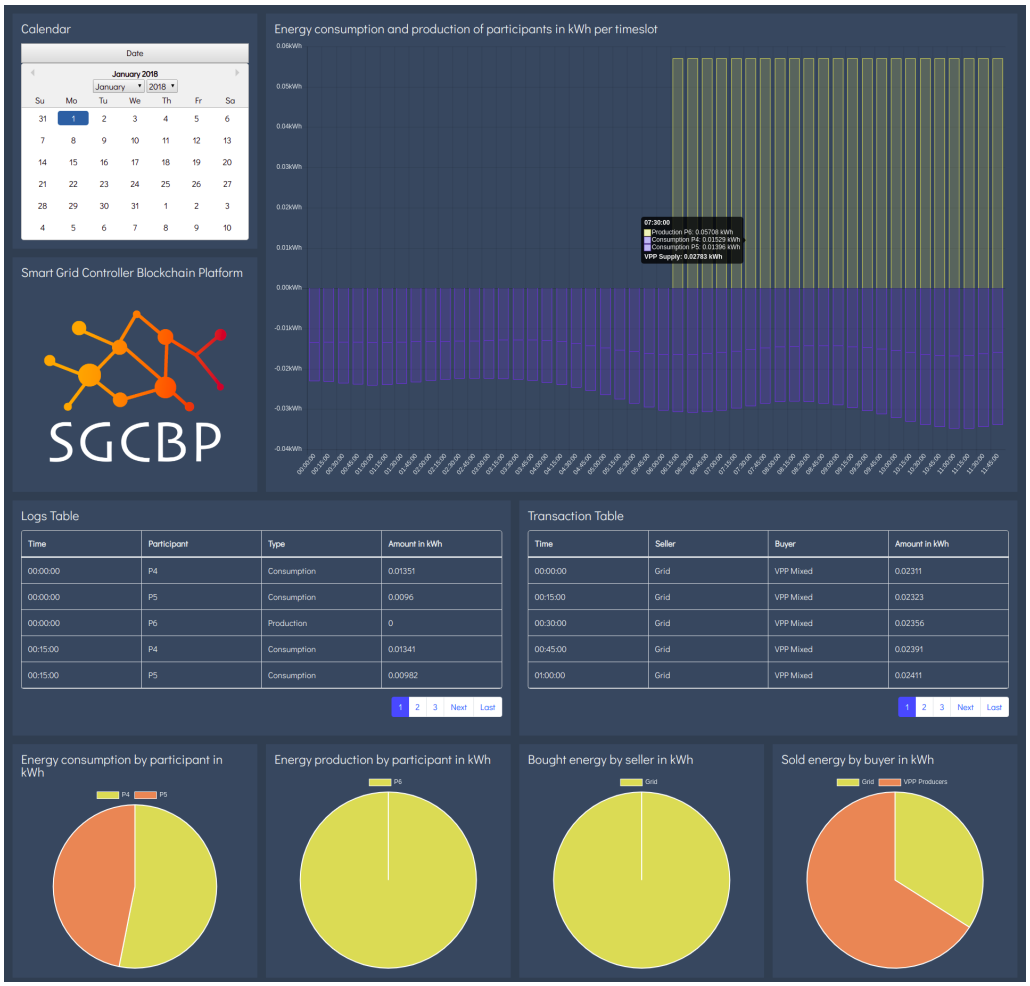

Figure 16: The figure shows the controller dashboard, which displays information about the VPP participant's energy production and consumption, and the VPP's transactions.

5.3.3.3 Master-Dashboard

The master dashboard, shown in Figure 17, displays information regarding all VPPs, like their logged energy surplus and demand data and transactions. The dashboard contains following components:

- Table component shows all transactions.
- Bar component represents the vpplogs of all VPPs.
- Two Pie components:
  - the first displays the amount of energy surplus per VPP, and
  - the second shows the amount of energy demand per VPP.

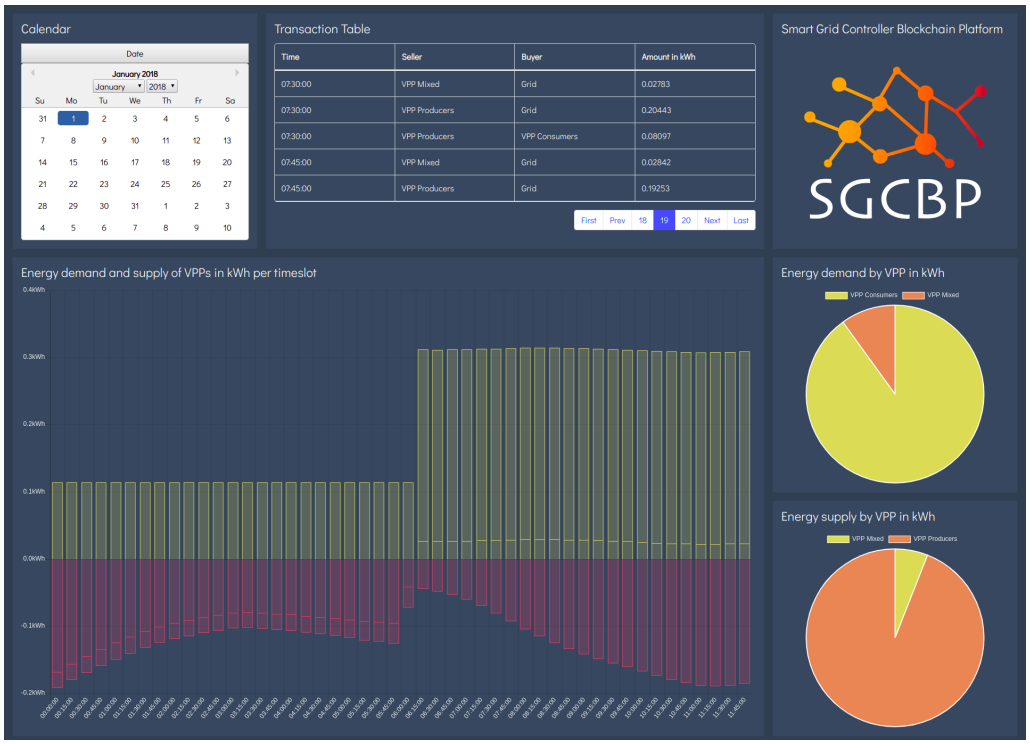

Figure 17: The figure shows the master dashboard, which displays information about the VPP's energy surpluses and demands, and their transactions.

5.3.4 Docker Images

To deploy the applications onto the RPIs, the applications were built into Docker images. Docker images are built from a Dockerfile, which contains all necessary instructions[84]. Each instruction creates a layer of the Docker image.

For all applications, the Docker image *node:8-alpine*[85], which is based on the Alpine Linux project[86], was used as base image. The biggest advantage of this image is the small size. Because all required node modules were installed by running the command `yarn install` as part of building the image, it was not needed to copy the directory *node_modules* to the image. Therefore, the file *.dockerignore* was created, which contains all directories and files, which are ignored by the build process.

To create an image the command `docker build` was executed. Thereby, the image was tagged by using the parameter `-t`, e.g., `docker build -t goranovic/participant-frontend`. The source code of the applications is available on the GitHub repository goranovic/sgcbp[69] and the images on the Docker Hub goranovic[61].

[84] Best practices for writing Dockerfiles `https://docs.docker.com/develop/develop-images/dockerfile_best-practices/` (accessed: January 8th 2019)

[85] nodejs/docker-node: Official Docker Image for Node.js `https://github.com/nodejs/docker-node` (accessed: January 8th 2019)

[86] index — Alpine Linux `https://alpinelinux.org/` (accessed: January 8th 2019)

5.4 Setup Of Raspberry Pi microSD Image

For the deployment of the platform, the RPIs had to be setup and configured. This means, for each RPI the operating system Raspbian needed to be flashed to a microSD card and all necessary components installed, such as NodeJS, Golang and Docker images. This section describes the process of setting up the Raspbian system and its components.

5.4.1 Raspberry Pi Basic Setup

The first step was to download the operating system Raspbian from the official repository[87]. Due to the support of graphical user interfaces (GUI), Raspbian Stretch with desktop was used in this thesis. To flash the downloaded image onto a microSD card, the tool Etcher[88] was used. After flashing the image, the microSD card was inserted into the RPI and the RPI powered on.

To activate the wireless network and Secure Shell access (SSH), the RPI configuration tool raspi-config[89] was used. After starting the tool, multiple menu items were shown, such as *Network Options*, *Interface Options* and *Boot Options*. The wireless network was configured in the Wi-fi options, located under the menu item *Network Options*. It required to enter the network service set identifier (SSID) and passphrase. For the thesis, a wireless network with the SSID *SGCBP* and passphrase *SGCBP1234* was used, which provides access to the internet. SSH was enabled with the menu *SSH* in the *Interfacing Options*. This eased the deployment process, by allowing remote access to each RPI. After an internet connection was established, existing packages were upgraded to new versions by using the Advanced Packaging Tool (APT)[90] and pip[91].

For the installation of Docker, the Docker convenience scripts[92] were used, which are meant for development environments. The script detects the OS and the version, and then installs the latest Docker version and all required dependencies and recommendations[93]. Further, to use Docker as standard user, which is not a root user, the standard user had to be added to the user group *docker*. Due to Kubernetes' missing swap support, the swap had to be disabled after Docker was installed[94]. Further, the memory and CPUset control groups (cgroups) had to be enabled for Kubernetes and Docker.

Next, Golang was downloaded from the official repository. The thesis used the Golang version 1.9.2 for the ARMv6 architecture[95]. To install the downloaded archive, it was extracted to the path */usr/local*. The paths to the extracted directory *go* and sub-directory *bin* were added to the environment variables `GOROOT` and `PATH`.

[87] Download Raspbian for Raspberry Pi `https://www.raspberrypi.org/downloads/raspbian/` (accessed: January 8th 2019)
[88] balenaEtcher - Home `https://www.balena.io/etcher/` (accessed: January 8th 2019)
[89] raspi-config - Raspberry Pi Documentation `https://www.raspberrypi.org/documentation/configuration/raspi-config.md` (accessed: January 8th 2019)
[90] apt(8) – apt – Debian stretch – Debian Manpages `https://manpages.debian.org/stretch/apt/apt.8.en.html` (accessed: January 8th 2019)
[91] pip - PyPI `https://pypi.org/project/pip/` (accessed: January 8th 2019)
[92] Docker convenience scripts `https://get.docker.com` (accessed: January 8th 2019)
[93] Get Docker CE for Ubuntu `https://docs.docker.com/install/linux/docker-ce/ubuntu/` (accessed: January 8th 2019)
[94] K8s on Raspbian `https://gist.github.com/alexellis/fdbc90de7691a1b9edb545c17da2d975` (accessed: January 8th 2019)
[95] Go v1.9.2 for ARMv6 `https://dl.google.com/go/go1.9.2.linux-armv6l.tar.gz` (accessed: January 8th 2019)

After Golang, NodeJS was installed by using the same procedure, as is to download the NodeJS archive and extract it to the path */usr/local*. For the thesis, the Node version 8.9.4 for the ARMv7 architecture[96] was used. The paths to the extracted directory *node* and sub-directory *bin* were added to the environment variables `NODEROOT` and `PATH`. As part of NodeJS, the javascript package manager Natural Polyglot Machine (NPM)[19], was installed. To prevent permission errors, the default NPM directory[97] was changed and added to the environment variable *PATH*.

Kubernetes can be installed by using APT. Therefore, the Kubernetes repository list had to be added to APT. Afterwards kubelet, kubectl and kubeadm were installed, for the thesis the version 1.9.6 was used. Further, with APT the command-line JSON processor jq[98] and the NFS kernel server[99] were installed.

As described in Section 4.1.3, each RPI is equipped with an OLED display. The simulations display messages on these displays, e.g., the participant simulation displays information about its energylogs. The used OLED display has four pins, which are connected to the RPI's GPIO pins. While the pins VCC and GND are required for the power supply, the pins SDA and SCL are used for the I2C interface. The detailed pin configuration is summarized in Table 17.

Table 17: The table contains the mapping of Raspberry Pi's GPIO pins to display pins.

OLED Pin	RPI GPIO Pin	Function
VCC	1	3.3 V power supply
GND	6	Ground
SCL	5	I2C Serial Clock
SDA	3	I2C Seria Data

By default, the I2C interface is disabled on the RPI. The I2C interface was enabled in raspi-config's *Interfacing Options*. Further, for the Node module rpi-oled the libraries i2c-tools and libi2-dev were installed.

To complete the RPI setup the Docker images of the HLF components and the platform applications were pulled from the Docker Hub goranovic. The complete setup script is available on the GitHub repository goranovic/sgcbp[69]. To ease the process of deployment, an image of the configured microSD card was created. In this way, the time-consuming process of installing the components on each RPI manually could be avoided. Only the pre-configured microSD image had to be flashed onto each RPI's microSD card and the configuration steps, which are described in the following section, had to be executed.

5.4.2 Configuration Steps After Flashing

After flashing the created image onto the microSD card, some steps were still necessary. First, the hostname, which was used as node identifier for the deployment, was changed to a unique hostname. This was done in the network options with raspi-config. For example, for the simulated

[96] Node v8.9.4 for ARMv7 `https://nodejs.org/download/release/v8.9.4/node-v8.9.4-linux-armv7l.tar.gz` (accessed: January 8th 2019)

[97] Resolving EACCES permissions errors when installing packages globally `https://docs.npmjs.com/getting-started/fixing-npm-permissions` (accessed: January 8th 2019)

[98] jq `https://stedolan.github.io/jq/` (accessed: January 8th 2019)

[99] Debian - Details of package nfs-kernel-server in stretch `https://packages.debian.org/en/stretch/nfs-kernel-server` (accessed: January 8th 2019)

environment the hostname of the RPI, containing peer1 of org1, was changed to org1peer1. Further, because the microSD image was used for each RPI, the *machine-id* needed to be changed. Otherwise, the communication in the Kubernetes cluster does not work, because the communication protocol have assigned the same MAC address to each RPI. This can be done by executing the commands in Code Snippet 15.

```
sudo rm /etc/machine-id
sudo systemd-machine-id-setup
cat /etc/machine-id
```

Code Snippet 15: The code snippet contains commands to change the *machine-id* for a Raspberry Pi.

The kubemaster acts as master of the Kubernetes cluster, which should not to be confused with the master of the platform. Before creating the cluster, the shared folder, containing files which are necessary for all RPIs, like connection profiles for the API applications and simulation datasets, had to be created and shared. Therefore, the nfs-kernel server package was used. The commands to share the shared folder, e.g., to the hosts of the IP network 192.168.0.0/24 are shown in Code Snippet 16.

```
sudo chmod ugo+rwx /shared
sudo chown nobody:nogroup /shared
"/shared 192.168.0.0/255.255.255.0(rw,sync,no_root_squash)" >> /etc/exports
sudo exportfs -ra
sudo service nfs-kernel-server restart
```

Code Snippet 16: The code snippet contains commands to setup the shared folder /shared.

The next step was to create the Kubernetes cluster, which was done on the kubemaster. Therefore, the commands in Code Snippet 17 had to be executed on the kubemaster, whose IP address in this example was set to 192.168.0.7.

```
sudo kubeadm init --token-ttl=0 --apiserver-advertise-address=192.168.0.7 --pod-
    ↪ network-cidr=10.244.0.0/16
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Code Snippet 17: The code snippet contains commands for the kubemaster to create the Kubernetes cluster.

After the creation of the cluster, a Kubernetes join command was outputted on the console. This command had to be executed on the other RPIs. An example is depicted in Code Snippet 18.

```
sudo kubeadm join --token 1a87d6.d70426bb4ce1686a 192.168.0.7:6443 --discovery-
    ↪ token-ca-cert-hash sha256:
    ↪ b5c1966591a72d3a1056fe6ca73a048521bb1fe93c6993fae6dab757a1b699a9
```

Code Snippet 18: The code snippet contains an example command to join the Kubernetes cluster.

Further, a communication protocol was required for the Kubernetes cluster. First, it was tried to use Weave[100], but this lead to RPI crashes. Therefore, the communication protocol was switched to flannel[101]. To install flannel as communication protocol for the Kubernetes cluster, the command in Code Snippet 19 was executed on the kubemaster.

```
curl -sSL https://rawgit.com/coreos/flannel/v0.9.1/Documentation/kube-flannel.
    ↪ yml | sed "s/amd64/arm/g" | kubectl create -f -
```

Code Snippet 19: The code snippet contains the command to setup flannel as communication protocol for the Kubernetes cluster.

In the meantime of the development of the SGCBP, the bug, regarding the use of Weave as communication protocol for the cluster, was fixed by its developers[102].

5.5 Smart Grid Controller Container Services

The next step was the deployment of the platform onto the RPIs. The IBM Container Services (ICS)[103] are normally used to launch an IBM Blockchain network on the IBM Blockchain Platform[104]. For the deployment onto the RPIs, a modified version of the ICS was created, called Smart Grid Controller Container Services.

According to the platform model in Section 4.3, a HLF network was built. For each VPP and for the Inter-VPP processes an organization and a channel was created. Regarding the HLF network entities, a participant's RPI hosts a peer and its CouchDB state database. The controller's RPIs host two peers and their CouchDB state databases, one for the Intra-VPP and one for the Inter-VPP organization. Further, the controller contains the CA for its Intra-VPP organization. On the master's RPI are a peer, its CouchDB state database, the CA for the Inter-VPP organization and the network's orderer located. An overview of this setting can be seen in Figure 18.

[100] Weave Net: Network Containers Across Environments — Weaveworks `https://www.weave.works/oss/net/` (accessed: January 8th 2019)
[101] flannel is a network fabric for containers, designed for Kubernetes `https://github.com/coreos/flannel` (accessed: January 8th 2019)
[102] Weave-Net Addon causing kernel panics on RPI 3B+. `https://github.com/weaveworks/weave/issues/3314` (accessed: January 8th 2019)
[103] IBM Blockchain Platform for Developers on IBM Container Service `https://github.com/IBM-Blockchain/ibm-container-service` (accessed: January 8th 2019)
[104] Getting Started with Blockchain. IBM. `https://www.ibm.com/blockchain/getting-started` (accessed: January 8th 2019)

Figure 18: The figure gives an overview of deployed HLF network entities and applications for the different platform nodes.

The creation of the HLF network involved the generation of the certificates and signing keys for the network entities and the configuration artifacts for the orderer and the channels [Hyp18]. Therefore, the tools cryptogen and configtxgen were used, which have been created as binaries. Cryptogen creates the cryptographic material, such as the certificates and keys, from the file *crypto-config.yaml*, which defines the orderer and peer organizations. It creates the directory crypto-config, which contains the certificates and keys. Configtxgen generates the configuration artifacts, e.g., the orderer genesis block and the channel configuration transactions, from the file *configtx.yaml*, which defines the network.

Before deploying the components to the RPIs, the Kubernetes cluster had to be active and the shared folder had to be populated. Therefore, following files and directories had to be located in the directory:

- chaincodes: the source code of the sgcbpintra and sgcbpinter chaincodes,
- configuration artifacts: the orderer genesis block and the channel configuration transactions, which have been generated with configtxgen,
- connection profiles: required by the APIs to connect to the HLF network,

- crypto-config: contains the certificates and keys for the HLF network entities, which have been generated with cryptogen, and
- datasets: all datasets for the simulation applications.

Further, as part of the deployment the directory hfc-key-store was created in the shared directory, which contains the certificates and key material, which are used by the APIs for interacting with the HLF network.

All components of the SGCBP, including the HLF network entities and the applications, were deployed by using the SGCCS. The SGCCS utilizes kubectl[105], which is a command line interface for managing Kubernetes clusters. It enables to create Kubernetes objects, like pods and services, by providing a YAML-file with object specifications.

The SGCCS requires a JSON configuration file to create all necessary object specifications. This JSON configuration file contains all necessary information for the deployment, like IP-adressess, hostnames, chaincodes and channels. The command line JSON-processor jq is used to process this JSON configuration file. The configuration data is filled into pre-defined YAML object specification templates for the HLF network entities and applications. The templates contain placeholders, which are replaced by the values from the configuration file by using the command sed[106]. The SGCCS is structured into the two directories kube-configs and scripts. While the directory kube-configs contains the object specifications and templates, all scripts have been placed into the directory scripts. To start the deployment, the script *createAll.sh* has to be executed.

In the following, the different steps of the deployment, visualized in Figure 19, are described. The source code of the SGCCS is available on the GitHub repository goranovic/sgcbp-container-services[107].

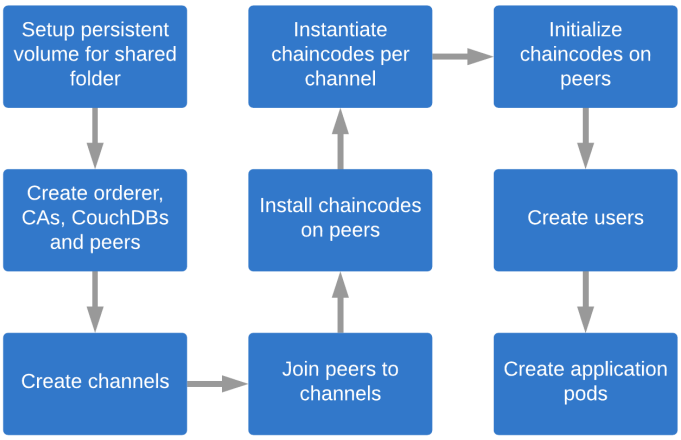

Figure 19: The figure shows the workflow of the deployment by using the SGCCS.

[105]Overview of kubectl - Kubernetes `https://kubernetes.io/docs/reference/kubectl/overview/` (accessed: January 8th 2019)
[106]sed, a stream editor `https://www.gnu.org/software/sed/manual/sed.html` (accessed: January 8th 2019)
[107]GitHub repository goranovic/sgcbp-container-services `https://github.com/goranovic/sgcbp-container-services` (accessed: January 8th 2019)

The first step is to integrate the shared folder into the cluster by creating a persistent volume and a persistent volume claim[108]. The persistent volume claim enables pods to use the persistent volume, in this case the shared folder, as volume and access its files. The second step is the deployment of the orderer, CAs, CouchDBs and peers. For each entity a pod and a service are created on the specified RPI.

The next steps require the Docker image fabric-tools, which provides a command line interface for the peers. The third step is the creation of the channels. Therefore, for each channel the command `peer channel create` is executed. As next, every peer has to join its channels. First the command `peer channel fetch config` is executed to retrieve the channel transaction block from the orderer and then the command `peer channel join` to join the channel.

As next, the chaincodes are installed on each peer. Before the installation, it has to be ensured that the chaincodes are available in the *GOPATH* of the container. To install a chaincode, the command `peer chaincode install` is required. After the chaincodes have been installed on all peers, they need to be instantiated. This must be done once per chaincode and channel by executing the command `peer chaincode instantiate`, which calls the method `Init` of the chaincode.

Chaincodes run in Docker containers on the RPIs. The first time a function of a chaincode is invoked, a Docker image for the chaincode is built. This process takes time and to prevent premature execution, as described in Section 5.2, the chaincode image should be built before the applications are started. Therefore, on each RPI the chaincodes have to be initially invoked to trigger the building of the chaincode image. This is done by using the command `peer chaincode query` and the function `Initialize` as parameter.

As described in Section 5.3.1, the API requires certificates and keys to interact with the HLF network. Therefore, for each organization an admin user is enrolled, which is used to register users for the applications. These users are then enrolled. Thereby the user's certificate and keys are stored in the directory hfc-key-store in the shared directory.

Finally, the application pods for the master, controllers, and participants are deployed. The deployed pods consist of the needed APIs, the simulation, and the dashboard. Services are created for the API and the dashboard, to make them accessible from other devices. This is necessary, because the Raspberry Pis are not connected to an external display and dashboards need to be called from another device, e.g., a computer or smartphone.

The object specifications for the simulation applications, which display data on connected OLED displays, differ slightly from the object specifications for other components. Due to the simulations accessing the GPIO pins, the simulation containers have to run in *privileged mode* and the pins need to be exposed to the containers[109].

After the SGCBP has been deployed to the RPIs, the pods and their status can be seen by executing the command `kubectl get pod --show-all` on the kubemaster. To get more details on a pod the command `kubectl logs <podname>` can be used, which prints the pod's container output. If a pod should contain more than one container, the name of the container has to be provided, as in `kubectl logs <podname> <containername>`. To open a dashboard in a browser, the IP address of the RPI hosting it and the port of the dashboard service need to be used.

[108] Persistent Volumes - Kubernetes `https://kubernetes.io/docs/concepts/storage/persistent-volumes/` (accessed: January 8th 2019)

[109] PiKube anyone? Yes please!! `https://imhotep.io/k8s/iot/2016/12/03/pikubee.html` (accessed: January 8th 2019)

5.6 Simulation

The SGCBP was tested by simulating the three VPPs: Consumers, Mixed and Producers. Each of them had three participants. For this setup, 14 RPIs were required:

- the kubemaster RPI
- one master RPI
- three controller RPIs
- nine participant RPIs

For the communication of these RPIs a wireless network with the SSID *SGCBP* was used, which was established by using a router. A photograph of the used setup is visible in Figure 20.

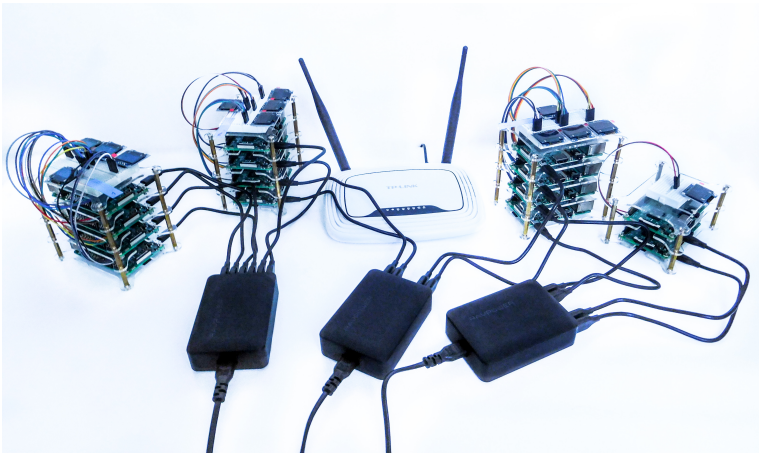

Figure 20: The figure shows the complete setup, including the router, the RPIs, the OLED displays and power sources.

The HLF network for the simulated environment consisted of four organizations, each with its own channel. Each RPI has been assigned a hostname corresponding to its organization and role, and for the deployment, a static IP address in the router settings by using MAC-IP bindings was assigned. This configuration is summarized in Table 18.

Based on this configuration, corresponding to Section 5.5 the files *crypto-config.yaml* and *configtx.yaml* for the creation of certificates and artifacts for the simulated environment were generated. Furthermore, following files were created for the deployment of the HLF network and SGCBP onto the RPIs:

- configuration artifacts generated with configtxgen,
- connection profiles for the API applications,
- cryptographic material generated with cryptogen,
- datasets for the simulation applications, and
- JSON configuration file for the SGCCS.

These files can be found on the GitHub repository goranovic/sgcbp-simulation[110]. After the shared folder had been populated with the required files, the deployment was executed.

Table 18: The table shows the simulation configuration, including the role, the hostname and IP address for each RPI.

VPP	Role	Pseudonym	Hostname	IP address
-	Kubemaster	-	kubemaster	192.168.0.7
-	Master	-	master	192.168.0.100
Producers	Controller	-	org1peer0	192.168.0.101
Producers	Part. Alpha	P1	org1peer1	192.168.0.111
Producers	Part. Bravo	P2	org1peer2	192.168.0.112
Producers	Part. Charlie	P3	org1peer3	192.168.0.113
Mixed	Controller	-	org2peer0	192.168.0.102
Mixed	Part. Delta	P4	org2peer1	192.168.0.121
Mixed	Part. Echo	P5	org2peer2	192.168.0.122
Mixed	Part. Foxtrot	P6	org2peer3	192.168.0.123
Consumers	Controller	-	org3peer0	192.168.0.103
Consumers	Part. Golf	P7	org3peer1	192.168.0.131
Consumers	Part. Hotel	P8	org3peer2	192.168.0.132
Consumers	Part. India	P9	org3peer3	192.168.0.133

5.7 Measurements

In this section, the platform operation is evaluated based on the displayed data in the dashboards. As part of the simulation, log files were created, which are used for evaluating the stability of the platform for the simulated environment. Further, to gain an insight on the deployment, the deployment time was measured.

5.7.1 Platform Operation

In this section, it is analyzed if the SGCBP works as expected by looking at the simulation results for the date *2018-01-01* and further by focusing on the timeslot *07:30*, which was chosen randomly. First, the displayed bar charts on the controller dashboards, which show a VPP's energy demand or energy supply course over a day, are analyzed. The participants of VPP Producers are only producers, which results for each timeslot in a VPP energy supply. The bar chart of the controller dashboard for this VPP shows only yellow bars, which represent timeslots with an existing energy surplus, as can be seen in Figure 21.

[110] GitHub repository goranovic/sgcbp-simulation `https://github.com/goranovic/sgcbp-simulation` (accessed: January 8th 2019)

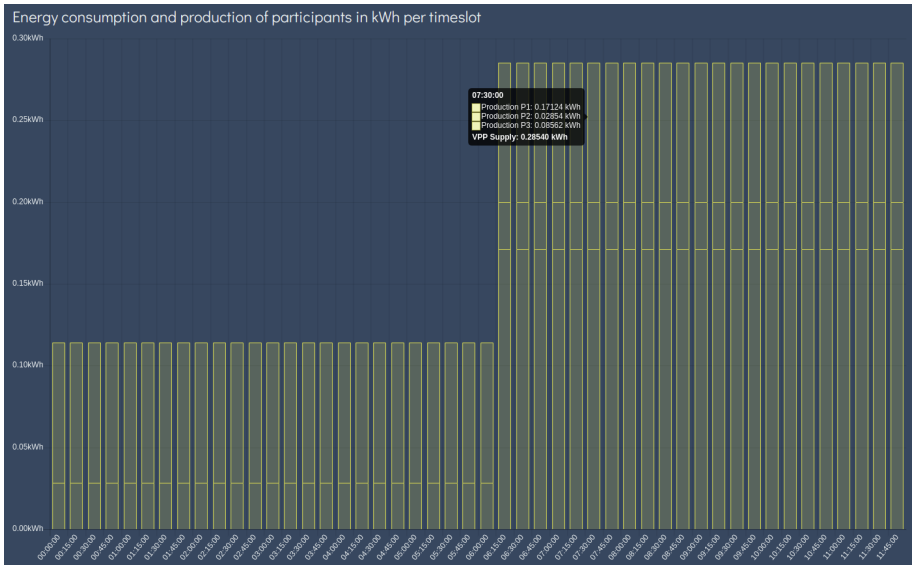

Figure 21: The figure shows the bar chart of the controller dashboard for VPP Producers with the hover box for the timeslot 07:30.

VPP Consumers consists only of consumers, which leads always to an energy demand. This is characterized in the controller dashboard, as shown in Figure 22, by only blue charts in the bar chart. These blue charts represent timeslots with an existing energy demand.

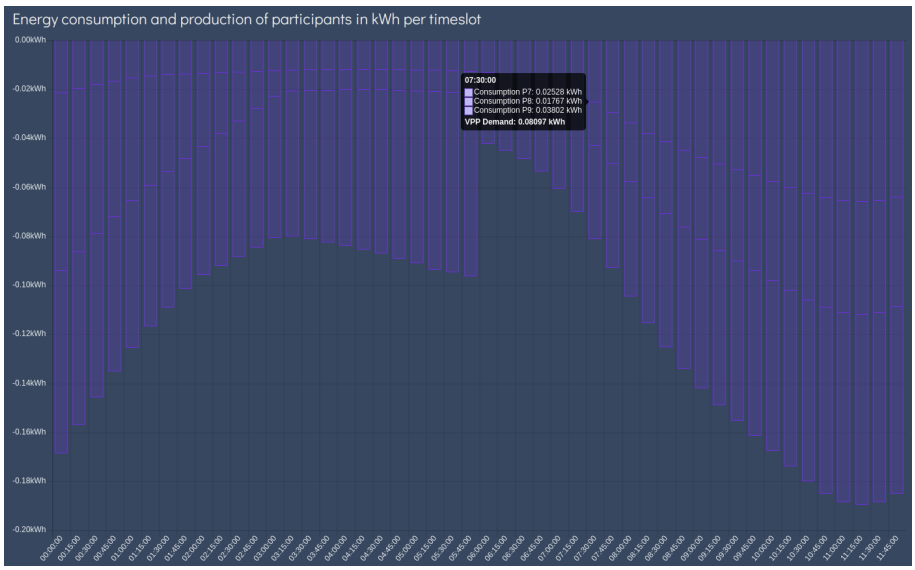

Figure 22: The figure shows the bar chart of the controller dashboard for VPP Consumers with the hover box for the timeslot 07:30.

In VPP Mixed consumers and producers are combined, which results into alternating energy demand and energy surplus timeslots. Thereby, the controller dashboard in Figure 23 shows a mix of blue and yellow bars in the bar chart.

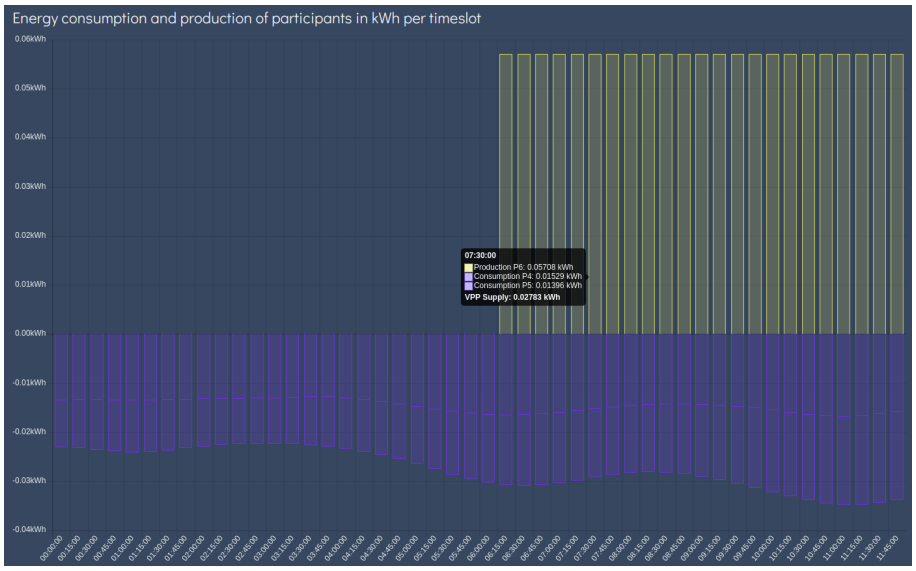

Figure 23: The figure shows the bar chart of the controller dashboard for VPP Mixed with the hover box for the timeslot 07:30.

As next the correctness of the displayed values should be proved for the timeslot 07:30. Therefore, the participant datasets for this timeslot are used as basis, which are summarized in Table 19 for VPP Producers, Table 20 for VPP Mixed, and Table 21 for VPP Consumers.

Table 19: The table shows the records of the participant datasets for the timeslot 07:30 of VPP Producers.

Participant	Type	Amount
Alpha	Production	0.17124
Bravo	Production	0.02854
Charlie	Production	0.08562

Table 20: The table shows the records of the participant datasets for the timeslot 07:30 of VPP Mixed.

Participant	Type	Amount
Delta	Consumption	0.01529
Echo	Consumption	0.01396
Foxtrot	Production	0.05708

Table 21: The table shows the records of the participant datasets for the timeslot 07:30 of VPP Consumers.

Participant	Type	Amount
Golf	Consumption	0.02528
Hotel	Consumption	0.01767
India	Consumption	0.03802

Based on these energylog data, the expected type and amount of the vpplogs for this timeslot have been determined, which are displayed in Table 22. These values correspond to the data in the hover boxes in Figure 24, which display the vpplog data for the chosen timeslot bar.

Table 22: The table shows the calculated vpplogs for the timeslot 07:30.

VPP	Type	Amount
VPP Producers	Supply	0.28540
VPP Mixed	Supply	0.02783
VPP Consumers	Demand	0.08097

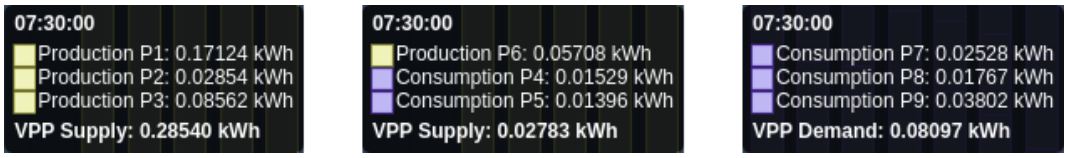

Figure 24: The figure shows the hover boxes of the controller dashboards for the timeslot 07:30.

The settlement should match these vpplogs according to the presented matching algorithm in Section 4.3.2 and create vpptransactions. Executing the settlement results in the creation of the vpptransactions in Table 23.

Table 23: The table shows the determined vpptransactions for the timeslot 07:30.

Seller	Buyer	Amount
VPP Producers	VPP Consumers	0.08097
VPP Producers	Grid	0.20433
VPP Mixed	Grid	0.02783

By comparing the vpptransactions in Table 23 and the created transaction in the simulation, which are shown in Figure 25, it can be deduced that the matching works proper.

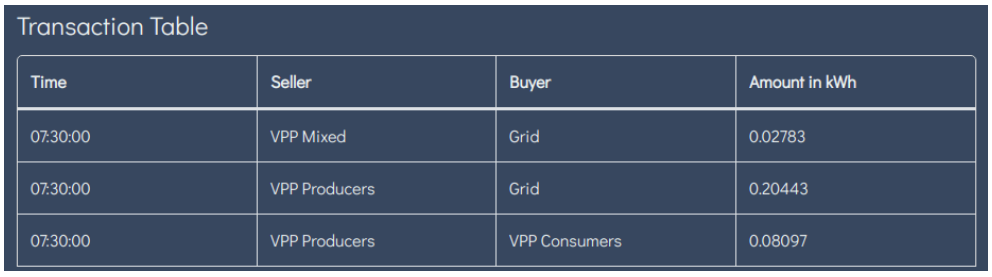

Figure 25: The figure shows the transaction table of the master dashboard with transactions for the timeslot 07:30.

5.7.2 Platform Stability

To measure the platform stability, log files were created by the simulation applications. These log files contain information about submitted transactions, e.g., about creating an energylog. Therefore, they consist of the simulated timeslot and the transaction status. A sample log file can be seen in Table 24.

Table 24: The table shows a sample log file of a participant.

Date	Time	Status
2018-01-01	00:00	success
2018-01-01	00:15	success
2018-01-01	00:30	eventhub down

Corresponding to the simulation datasets, each log file contains 240 entries. The total number of log file entries is 3120 with 114 errors, which is an error rate of 3.65 %. All errors are of the same type, which is *eventhub down*. This indicates that the eventhub, which is responsible to inform an application about the transaction commitment to the ledger, was not reachable. Nevertheless, the transactions have been committed to the ledger, only the applications have not been notified. The source of these eventhub errors remains unknown. The occurrence of multiple eventhub errors for one timeslot indicates that network errors could be responsible. The log file entries with status *success* indicate that these have been written successfully to the ledger.

5.7.3 Deployment Process

The deployment process was evaluated by measuring its execution time. Therefore, the line in Code Snippet 20 was added to the script *createAll.sh* of the SGCCS, which executes the deployment process. This instruction prints additionally the execution timestamp for each line of the script.

```
PS4='+ $(date "+%s.%N")'
```

Code Snippet 20: The code snippet shows the instruction to display the execution time per line.

Based on the timestamps the execution times of the deployment steps were calculated, which are shown in Table 25. The complete deployment took 4,017.09 seconds, whereby the setup of chaincodes, including installation, instantiation and initialization, was responsible for 79.94 % of the execution time.

Table 25: The table contains the execution times of the deployment process steps in seconds.

Step	Duration in seconds
Setup storage	12.06
Create orderer, CAs, CouchDBs and peers	154.32
Create channels	64.14
Join channels	266.06
Install chaincodes	953.26
Instantiate chaincodes	523.65
Initialize chaincodes	1,734.22
Register and enroll users	186.31
Create applications	123.07

5.8 Simulation Results

As can be concluded from the platform operation analysis, the SGCBP behaves as expected. The bar chart figures of the controller dashboards display the expected shapes of the energy demand and supply for the different VPPs. Furthermore, the transaction table in the master dashboard listens the expected vpptransactions. This shows that the creation of the vpplogs based on recorded energylogs, and the settlement of the vpplogs works proper.

Although, the simulation logs show some errors, these all are eventhub errors, which do not have an impact on the platform operation. Therefore, it can be concluded that the platform operates stable for the simulated environment. Nevertheless, it has to be taken into account that for a production environment the number of participants and controllers would be much higher. Further, another timescale would be used, because the simulation compresses fifteen minutes into fifteen seconds. An extended timeslot would increase the available time for the Intra- and Inter-VPP processes, and the transaction ordering service. For example, by increasing the time for the settlement process a higher number of VPPs could be integrated on the platform. Currently, the platform integrates a SOLO orderer as ordering service. In a production environment deploying a distributed ordering service, e.g., Kafka, the transaction processing is more complex and takes more time. Therefore, in future work it is necessary to further evaluate the platform in combination with a distributed ordering service and growing numbers of participants.

The measurements of the deployment process show that most time is consumed by setting up the chaincodes. Especially, the building of the chaincode images, which takes place during instantiation and initialization of the chaincodes, consumed 58.3 % of the deployment time. Several deployment steps target the RPIs separately, such as the building of the chaincode images. These steps should be parallelized, as far as possible, which would reduce the deployment time.

For example, by using the maximum execution times for the chaincode instantiation and initialization, which can be seen in Table 26, instead of the execution times in Table 25, to calculate the deployment time, a deployment time of 2,230.44 seconds is calculated. This indicates that by only parallelizing the building of the chaincode images, the execution time of the deployment process could be reduced by 44.47 %.

Table 26: The table contains the maximum execution times of the chaincode instantiation and initialization in seconds.

Step	Duration in seconds
Instantiate chaincodes	141.46
Initialize chaincodes	329.76

6 Discussion & Outlook

This chapter starts with a brief summary of the thesis, focusing on the developed SGCBP and simulation. An outlook on Hyperledger Fabric and blockchain technology in regard to the GDPR follows. Based on this, a model and recommendations for a future version of the platform are presented. Finally, an outlook and vision for smart energy systems in regard to blockchain technology are given.

6.1 Summary

The thesis presents a blockchain-based Smart Grid Controller for local and regional consumer driven decentralized virtual power plants. It provides an overview of VPPs, smart grids, GDPR and privacy design strategies, blockchain technology and multiple blockchain implementations. For the comparison of these blockchain implementations, multiple criteria, such as light energy-footprint and integration of privacy mechanisms, have been deduced from various Smart Grid Controller platform requirements. Based on the comparison, HLF presented itself as most suitable blockchain implementation for the development of the SGCBP.

For a generic VPP environment, a platform model has been developed. Thereby, the platform differentiates Intra-VPP from Inter-VPP processes. Intra-VPP processes include the logging of energy production and consumption by the participant and the calculation of the VPP's energy surplus and demand. Inter-VPP processes focus on the reported energy surplus and demand of all VPPs. They are matched together according to the presented matching algorithm. Further, the platform model enhances privacy by integrating pseudonyms and encryption.

The first step in the development of the platform was to port the Hyperledger Fabric components onto the Raspberry Pi 3B. Therefore, the source code of HLF v1.1 was adapted to suit the ARMv7 architecture of the RPI3B. For the platform processes, e.g., protocolling and matching, the chaincodes sgcbpintra and sgcbpinter have been created. These implement methods to query the ledger or to create energylogs, vpplogs, and vpptransactions. As part of the platform, API, dashboard, and simulation applications have been created. The APIs provide endpoints to connect to the HLF network. The dashboards display ledger data in different forms, such as pie and bar charts. The simulations are used to simulate the master, the controllers and the participants of the VPP environment.

Encryption is applied in the chaincode sgcbpintra and in the applications. Thereby, AES-256 in CBC-mode is used. This double-encryption in chaincode and applications guarantees, that if an

attacker should get knowledge of one encryption key, they would not be able to decrypt the data without the other encryption key.

The setup of the created demonstration and simulation hardware is described in detail. The descriptions include the necessary configuration steps for the operating system Raspbian, including the installation of necessary software packages and their configuration. To simplify the setup process, the creation of a microSD image has been described, which includes a pre-configured operating system. After this image is flashed onto a microSD card, required post-configuration steps are described.

The thesis describes the SGCCS, which was created for the deployment of the platform. It consists of a modified version of the IBM Container Services. Based on a configuration file, the HLF network and the platform applications are deployed onto the Rasbperry Pis.

For testing the platform, an environment consisting of three VPPs, each with three participants, was simulated. Therefore, the SGCCS and developed applications were used. The deployment process and the stability of the SGCBP in the simulation environment, have been evaluated. For the small number of platform nodes in the simulation environment, the platform operates stable. The deployment of the platform in the simulated environment takes around 70 minutes. This could be improved by utilizing parallelization. First calculations indicate, that by parallelizing the building of chainode images on the RPIs, the execution time can almost be cut in half.

The complete code of the thesis can be found on the following GitHub repositories:

- goranovic/fabric[62]
- goranovic/fabric-ca[63]
- goranovic/sgcbp[69]
- goranovic/sgcbp-container-services[107]
- goranovic/sgcbp-simulation[110]

All created Docker images are located on the Docker Hub goranovic. The created HLF components and the SGCCS can be used to deploy a HLF network onto a Kubernetes cluster of Raspberry Pis. Further, the SGCBP includes all necessary applications, such as API for connection to the HLF network, dashboard for querying the ledger and visualizing the data, and simulations for writing data to the ledger. Now, the platform can be used as basis to gather knowledge about the setup of blockchain applications in the smart grid domain and for the development of these.

6.2 Future Development With Hyperledger Fabric

The progress in the development of HLF seems very promising. Some features, which have been added since HLF version 1.1, are:

- Private Data Collections to keep data confidential between channel members,
- Service Discovery for orderers, peers, chaincode and endorsement policies,
- Chaincode written in Java is supported, or
- Identity Mixer enables anonymous and unlinkable identities by using zero-knowledge proofs.

Still, there is space for improvements, e.g., with the private data feature, introduced in HLF v1.2. While data can be kept confidential, this is limited to organization-level. This means, that it is not possible to keep data confidential between specific members in the same organization. Another feature, which is planned, is the possibility of a chaincode calling another chaincode on a different channel to write data[111]. Regarding the platform, this feature could be used, e.g., to integrate a part of the controller functionality directly into the chaincode.

The development of HLF networks and applications could be improved by the creation of a HLF development toolchain. For example, the Hyperledger Composer eases the development of HLF applications. But it is only recommended for proof-of-concepts, due to the lack of features, as the support of multiple channels[64].

Officially HLF does not support the ARM architecture. While creating this thesis, there has been another attempt to get HLF onto the RPI[112]. This attempt focused on the Raspberry Pi 3B's support of 64-bit operating systems. Although the official operating system Raspbian is only a 32-bit OS, there exist experimental 64-bit OS for the RPI, like pi64[113]. As part of this attempt, Docker images of the HLF components have been created for the AARCH64 architecture, which are available on the Docker Hub pesicsasa[114].

6.3 Discussion Of Blockchain And The General Data Privacy Regulation

As presented in Section 2.4.3, solutions to ensure privacy and data protection by design on the blockchain, such as state channels and zero knowledge proofs, are being developed. Regarding the GDPR, a lot of challenges still exist for the development of blockchain applications. On October 16th 2018 the European Union Blockchain Observatory & Forum released a report [OF18] focusing on the relationship between blockchain technology and the GDPR. The authors identified three main issues:

- Identification and obligations of data controllers and processors: In contrast to centralized systems, the identification of data controllers and processors in decentralized systems, such as blockchain networks, is difficult. This is especially the case for public, permissionless blockchain networks, where everyone can be a validating and participating node.
- Anonymization of personal data: Applications storing personal data, have to comply to the GDPR. For anonymized data the GDPR does not apply. Therefore, an option would be to anonymize personal data and store it on the blockchain. The problem thereby is, that there does not exist a consensus on when data is sufficiently anonymized to be stored on the blockchain. On one hand, anonymized data should not be reversable to the personal data. For example, only hashes of the personal data could be stored on the blockchain and the personal data stored off-chain. Still hashes could be reversed to personal data with brute force attacks. It cannot be guaranteed that current encryption techniques will not be cracked in the future, e.g., by quantum computers. On the other hand, anonymized data should not be linkable to personal data, e.g., by using pattern analysis.

[111] Chaincode calling chaincode `https://jira.hyperledger.org/browse/FAB-1788` (accessed: January 8th 2019)
[112] Hyperledger Fabric for ARM `https://jira.hyperledger.org/browse/FAB-10382` (accessed: January 8th 2019)
[113] A 64-bit OS for the Raspberry Pi 3 `https://github.com/bamarni/pi64` (accessed: January 8th 2019)
[114] pesicsasa's Profile - Docker Hub `https://hub.docker.com/u/pesicsasa` (accessed: January 8th 2019)

- Excercise of data subject rights: The GDPR empowers data subjects with multiple rights, such as the *Right to Access* and *Right to be Forgotten*. A main challenge for the blockchain technology is the erasure of data from the blockchain, whose key feature is its immutability. Because the GDPR does not define, what *erasure of data* means, different approaches have been proposed, like key-destruction, which may not comply to a strict interpretation of the GDPR. In key-destruction, data is stored encrypted on the blockchain and instead of erasing the data, the encryption key is deleted. Thereby, the data remains encrypted on the blockchain and cannot be decrypted anymore.

Because of that the authors propose four rule-of-thumb principles for developing applications, which implement blockchain technology[OF18]:

- Evaluate if the blockchain is needed.
- Do not store personal data, or use privacy design strategies to anonymize it.
- Store personal data off-chain or on private, permissioned blockchains.
- Innovate and provide transparency to users.

While these principles are a good guideline for the development of blockchain applications, nevertheless improvements are needed, especially with regard to public, permissionless blockchains. Decentralized networks, like blockchain networks, have not been in the focus when the GDPR was created. Further, the GDPR is missing important definitions, e.g., the *erasure of data*. Therefore, the GDPR can be revised or supplemented with the involvement of lawyers, regulation bodies and experts from the blockchain scene. Although, a hype around blockchain technology exists, it is still in its early stages. There are a lot of challenges, which need to be solved, such as the detection of breaches.

6.4 Model For A Future Platform Version

For the development of a next platform version there exist multiple ideas, which are presented in this section. These ideas focus on several aspects, covering platform architecture, security and testing.

The developed Smart Grid Controller Blockchain Platform assumes that the energy production or consumption is aggregated at participant level and only the difference is stored on the platform. For a next version, the platform should be extended to include processes inside buildings. This means, the participants should have the opportunity to log their energy production and consumption at device level. This data should be kept private to the participant and stored in a local database. Currently each RPI is equipped with a small OLED display. This OLED display should be replaced by a larger touchscreen. The device-level information could be processed and displayed on the participant's dashboard, for example, the participant's energy consumption per device. This could raise the participant's awareness regarding their energy production and consumption. Thereby, a middleware application could be responsible to synchronize the database with the ledger. This means, it would aggregate the participant's energy production and consumption, and save the difference as energylog on the ledger. Additionally, the energylog could be stored in the local database. The participant's dashboard would only have to access the database to retrieve necessary information. This would increase the performance of the dashboard and reduce the number of requests to the HLF network. The model of this setup is shown in Figure 26.

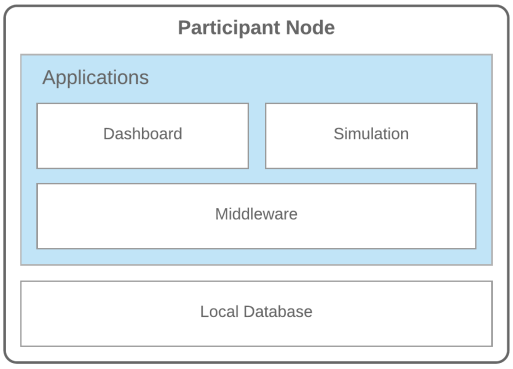

Figure 26: The figure shows the participant architecture of the future platform model.

To trigger the settlement the SGCBP requires a master, which represents a central instance. This should be avoided, and it should be evaluated how the master could be replaced by a distributed solution. An option would be to use controllers for triggering the settlement. Thereby, a policy could be enforced, e.g., that more than 50 % of the controllers have to agree to the triggering.

For the proof-of-concept a single SOLO ordering service has been applied. The failing of this centralized ordering service would have major implications on the HLF network. For a production environment, the ordering service should be changed to a distributed fault-tolerant ordering service, e.g., the recommended Kafka ordering service. Depending on the number of participants and transactions to order, the timeframe has to be evaluated for these distributed solutions.

A necessary step towards a market-ready product, is the compliance to the GDPR and the implementation of additional privacy design strategies, especially process-oriented strategies. Therefore, extending privacy mechanisms to controllers should be considered. Some questions, that should be clarified are e.g.:

- Should pseudonyms be used for VPPs?
- Do vpplogs have to be encrypted?
- Do vpptransactions have to be encrypted?
- Is it necessary to keep transactions confidential between VPPs?

Further, it should be considered using asymmetric encryption, thereby enhancing security and empowering the participant. For the proof-of-concept, private keys for the HLF network are stored in PEM-encoded files. This is a security issue, which can be solved by using supported Hardware Security Modules (HSM)[115]. Especially, if considering the use of RPIs, where all files are stored on a microSD card. This microSD card could be removed and cloned. Therefore, the RPIs should be equipped with a HSM to manage private keys.

Chaincodes should implement Attribute-Based Access Control (ABAC)[115] and use an attribute `role` for access control. Each user certificate would contain this attribute `role`, indicating if the user is a participant or controller. This enables to restrict access to chaincode functions by role, e.g., querying all energylogs should only be allowed to controllers. Additionally, instead of using

[115]Fabric CA User's Guide `https://hyperledger-fabric-ca.readthedocs.io/en/latest/users-guide.html` (accessed: January 8th 2019)

cryptogen to create the required certificates for the HLF network, a production-ready solution, like the Hyperledger Fabric CA should be utilized. Further to enhance security, the HLF network should use Transport Layer Security (TLS) [Hyp18]. TLS would provide secure communication and client authentication by using TLS handshakes.

As operating system the 32-bit Raspbian was used for the proof-of-concept. It should be evaluated if the OS should be switched to a 64-bit OS, due to the partial support of HLF for 32-bit operating systems. A switch would enable to use newer HLF versions without the need for extensive modifications.

For the proof-of-concept Docker was installed by using the Docker convenience scripts, as described in Section 5.4.1. These scripts are only meant for development environments and not for production. Therefore, Docker should be installed by using APT and the official Docker Hub[93]. Furthermore, an issue regarding Docker which was encountered, was the disappearance of multiple Docker images on the RPIs. A pattern for occurrence and the source of this problem could not be identified. This issue has to be analyzed thoroughly and solved for production environments.

The simulation applications are developed to run only one hour, to run longer simulations these would need to be adapted, and corresponding datasets provided. In a production environment instead of the participant simulations, devices, e.g., smart meters, would log their energy consumption or production, and the simulation would not be restricted to a specific duration. This would require to modify the controller and master simulations, and integrate mechanisms for HLF entities, e.g., to extend the memory for the peers with growing ledger size.

As already suggested in Section 5.8, the deployment process should be improved by parallelization. Another essential improvement would be to use hostnames instead of hardcoded IP addresses. This would allow to reach some degree of abstraction between the Kubernetes cluster and the SGCBP, but would require the integration of a DNS, like Kube-DNS[116].

Alternatively, instead of hosting the HLF network on the RPIs, the HLF network could be hosted in the cloud. For example, the IBM Cloud Blockchain Platform provides HLF as a service. This separation of the HLF network and the applications, would enable to use devices with lower requirements to performance and thereby reduce costs. A suitable candidate is e.g., the Raspberry Pi Zero W, which could be used for the local applications inside future Smart Grid Control devices.

For in-depth testing of HLF networks several different tools exist, like fabric-test[117], Hyperledger Caliper[118] and Gauge[119]. A next platform version should integrate one of these. Therefore, the mentioned tools should be evaluated and compared.

6.5 Outlook & Vision For Smart Energy Systems

Regulations will play an important role in the adoption of blockchain technology. Not only data regulations, such as the GDPR, are of importance, but also technical regulations. The regulation of emerging technologies, e.g., the blockchain technology, is a complex and time-consuming

[116] DNS for Services and Pods - Kubernetes `https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/` (accessed: January 8th 2019)

[117] Welcome to fabric-test `https://github.com/hyperledger/fabric-test` (accessed: January 8th 2019)

[118] Hyperledger Caliper - Blockchain performance benchmarking for Hyperledger Burrow, Fabric, Iroha & Sawtooth `https://hyperledger.github.io/caliper/` (accessed: January 8th 2019)

[119] Gauge - Performance benchmarking tool for Hyperledger Fabric and Quorum `https://github.com/persistentsystems/gauge` (accessed: January 8th 2019)

process. It requires the participation of several interest groups of the government, lawyers, transmission system operators, distribution system operators, developers, and end users. Nevertheless, regulations should not be obstacles for the development of beneficial blockchain applications.

Reaching consensus is a crucial element of blockchain networks. Different consensus algorithms exist, Proof-of-Work or Proof-of-Stake to name a few. In case of energy systems, the consensus algorithm should fulfill several requirements such as a respectfully low energy consumption, or scalability. Incentive mechanisms for validating nodes are a related but often ignored aspect. Validating nodes, which are responsible for the consensus in blockchain networks, require processing power and memory to process and store the ledger. This produces costs, which need to be compensated. Therefore, it is important to give an incentive to operate a validating node. For example, Bitcoin does this by offering transaction fees to reward miners.

Regarding energy systems, which are highly critical infrastructure, a cyberattack could cause severe consequences, e.g., damage equipment or endanger people's lives. Therefore, special focus needs to be put on cybersecurity. Besides integrating permissioning, it has to be thought of mechanisms to detect cyberattacks and limit their scope. Furthermore, it needs to be specified how a system should act when errors occur. This is especially challenging due to blockchain's immutability, which is in contrast to reversing or deleting a transaction.

The variety of blockchain implementations, such as Ethereum and Hyperledger Fabric, leads to new challenges, for example, interoperability. Another option would be to use blockchain implementations, which focus on the energy sector, e.g., the Energy Web Foundation's Energy Web Chain. The Energy Web Chain is based on Ethereum but uses a Proof-of-Authority consensus algorithm, called Aura. Furthermore, the Energy Web Chain integrates smart contracts, private transactions, and a permissioning system to comply with regulations and requirements of the energy sector.

Currently, a hype around blockchain and distributed ledger technologies does exist. It generates a perception, where blockchain technology seems to be the holy grail for decentralized systems by providing a decentralized trustless system to store and process information. The number of projects, which work on solutions and applications based on blockchain technology is rising, but one needs to be aware, that some projects only misuse the existing blockchain hype for marketing purposes.

Although blockchain technology and proof-of-concepts have been bubbled up for some years, the realization of applications has just begun. Before blockchain technology can reach its full potential, it will need to mature and overcome several challenges, e.g., technological, but also regulatory problems. Therefore, it is difficult to predict, which impact it will have on the energy sector and when. As most projects focus on P2P energy trading, these applications will likely be the first to be market-ready. Nevertheless, other use cases should not be ignored, especially regarding grid management, which is of interest for grid operators.

List of Figures

List of Tables

List of Code Snippets

Literature

[BCGH16] Brown, Richard G. ; Carlyle, James ; Grigg, Ian ; Hearn, Mike: Corda: An introduction. In: *R3 CEV, August* (2016)

[BKR11] Bogdanov, Andrey ; Khovratovich, Dmitry ; Rechberger, Christian: Biclique cryptanalysis of the full AES. In: *International Conference on the Theory and Application of Cryptology and Information Security* Springer, 2011, S. 344–371

[C+95] Commission, European C. [u. a.]: Directive 95/46 EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and the free movement of such data. (1995)

[CCMN17] Castellanos, Alejandro ; Coll-Mayor, Debora ; Notholt, Antonio: Cryptocurrency as guarantees of origin: simulating a green certificate market with the ethereum blockchain. (2017)

[CD16] Christidis, Konstantinos ; Devetsikiotis, Michael: Blockchains and smart contracts for the internet of things. In: *Ieee Access* 4 (2016), S. 2292–2303

[CF18] Climate ; Fund, Energy: Climate and Energy Model Regions – An Austrian blueprint for a successful bottom-up approach in the field of climate change and energy. (2018). – `https://www.klimaundenergiemodellregionen.at/assets/Uploads/Publikationen/2018-Fact-Sheet-Climate-and-Energy-Model-Regions-EN-final.pdf` (accessed: January 8th 2019)

[Cou14] of the Council, General S.: European Council (23 and 24 October 2014) – Conclusions. In: *EUCO 169/14* (2014). – `https://www.consilium.europa.eu/uedocs/cms_data/docs/pressdata/en/ec/145397.pdf` (accessed: January 8th 2019)

[DSGI+18] Di Silvestre, Maria L. ; Gallo, Pierluigi ; Ippolito, Mariano G. ; Sanseverino, Eleonora R. ; Zizzo, Gaetano: A Technical Approach to the Energy Blockchain in Microgrids. In: *IEEE Transactions on Industrial Informatics* 14 (2018), Nr. 11, S. 4792–4803

[Dwo01] Dworkin, Morris: Recommendation for block cipher modes of operation. methods and techniques / NATIONAL INST OF STANDARDS AND TECHNOLOGY GAITHERSBURG MD COMPUTER SECURITY DIV. 2001. – Forschungsbericht

[EC] E-Control: Electricity Market Code - Chapter 6 - Meter Readings, Data Formats and Standardised Load Profiles. . – `https://www.e-control.at/documents/20903/-/-/90219f4d-ba76-41d6-948f-3d82c82a42e3` (accessed: January 8th 2019)

[EMST76] Ehrsam, William F. ; Meyer, Carl H. ; Smith, John L. ; Tuchman, Walter L.: Message verification and transmission error detection by block chaining. (1976). –

US Patent 4,074,066

[EU15] of Justice of the European Union, Court: The Court of Justice declares that the Commission's US Safe Harbour Decision is invalid. In: *PRESS RELEASE No 117/15* (2015). – `https://curia.europa.eu/jcms/upload/docs/application/pdf/2015-10/cp150117en.pdf` (accessed: January 8th 2019)

[Far10] Farhangi, Hassan: The path of the smart grid. In: *IEEE power and energy magazine* 8 (2010), Nr. 1

[FMXY12] Fang, Xi ; Misra, Satyajayant ; Xue, Guoliang ; Yang, Dejun: Smart grid-The new and improved power grid: A survey. In: *IEEE communications surveys & tutorials* 14 (2012), Nr. 4, S. 944–980

[GMF+17] Goranović, Andrija ; Meisel, Marcus ; Fotiadis, Lampros ; Wilker, Stefan ; Treytl, Albert ; Sauter, Thilo: Blockchain applications in microgrids an overview of current projects and concepts. In: *Industrial Electronics Society, IECON 2017-43rd Annual Conference of the IEEE* IEEE, 2017, S. 6153–6158

[Hea16] Hearn, Mike: Corda: A distributed ledger. In: *Corda Technical White Paper* (2016)

[Hoe14] Hoepman, Jaap-Henk: Privacy design strategies. In: *IFIP International Information Security Conference* Springer, 2014, S. 446–459

[HP17] Halpin, Harry ; Piekarska, Marta: Introduction to Security and Privacy on the Blockchain. In: *Security and Privacy Workshops (EuroS&PW), 2017 IEEE European Symposium on* IEEE, 2017, S. 1–3

[HSLC17] Hahn, Adam ; Singh, Rajveer ; Liu, Chen-Ching ; Chen, Sijie: Smart contract-based campus demonstration of decentralized transactive energy auctions. In: *Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT), 2017 IEEE* IEEE, 2017, S. 1–5

[HXWL18] Huang, Xiaohong ; Xu, Cheng ; Wang, Pengfei ; Liu, Hongzhe: Lnsc: A security model for electric vehicle and charging pile management based on blockchain ecosystem. In: *IEEE Access, vol. PP* (2018), Nr. 99, S. 1–1

[Hyp18] Hyperledger: hyperledger-fabricdocs Documentation – Release master. (2018). – `https://media.readthedocs.org/pdf/hyperledger-fabric/release-1.1/hyperledger-fabric.pdf` (accessed: January 8th 2019)

[KYH+17] Kang, Jiawen ; Yu, Rong ; Huang, Xumin ; Maharjan, Sabita ; Zhang, Yan ; Hossain, Ekram: Enabling localized peer-to-peer electricity trading among plug-in hybrid electric vehicles using consortium blockchains. In: *IEEE Transactions on Industrial Informatics* 13 (2017), Nr. 6, S. 3154–3164

[LBA17] Lundqvist, Thomas ; de Blanche, Andreas ; Andersson, H Robert H.: Thing-to-thing electricity micro payments using blockchain technology. In: *Global Internet of Things Summit (GIoTS), 2017* IEEE, 2017, S. 1–6

[LFP08] Lugmaier, Andreas ; Fechner, Hubert ; Pruggler, Wolfgang: National technology platform-smart grids austria. (2008)

[LKY+18] Li, Zhetao ; Kang, Jiawen ; Yu, Rong ; Ye, Dongdong ; Deng, Qingyong ; Zhang, Yan: Consortium blockchain for secure energy trading in industrial internet of things. In: *IEEE transactions on industrial informatics* 14 (2018), Nr. 8, S. 3690–3700

[LR17] Lazaroiu, Cristian ; Roscia, Mariacristina: Smart district through IoT and Blockchain. In: *Renewable Energy Research and Applications (ICRERA), 2017 IEEE 6th International Conference on* IEEE, 2017, S. 454–461

[MMM17] Münsing, Eric ; Mather, Jonathan ; Moura, Scott: Blockchains for decentralized

optimization of energy resources in microgrid networks. In: *Control Technology and Applications (CCTA), 2017 IEEE Conference on* IEEE, 2017, S. 2164–2171

[MPM17] Mannaro, Katiuscia ; Pinna, Andrea ; Marchesi, Michele: Crypto-trading: Blockchain-oriented energy market. In: *AEIT International Annual Conference, 2017* IEEE, 2017, S. 1–5

[Nak08] Nakamoto, Satoshi: Bitcoin: A peer-to-peer electronic cash system. (2008)

[OF18] Observatory, European Union B. ; Forum: Blockchain and the GDPR. (2018). – `https://www.eublockchainforum.eu/sites/default/files/reports/20181016_report_gdpr.pdf` (accessed: January 8th 2019)

[OST06] Osvik, Dag A. ; Shamir, Adi ; Tromer, Eran: Cache attacks and countermeasures: the case of AES. In: *Cryptographers' Track at the RSA Conference* Springer, 2006, S. 1–20

[PDVBD15] Plancke, Glenn ; De Vos, Kristof ; Belmans, Ronnie ; Delnooz, Annelies: Virtual power plants: Definition, applications and barriers to the implementation in the distribution system. In: *European Energy Market (EEM), 2015 12th International Conference on the* IEEE, 2015, S. 1–5

[PKS16] Pustisek, Matevz ; Kos, Andrej ; Sedlar, Urban: Blockchain Based Autonomous Selection of Electric Vehicle Charging Station. In: *2016 International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI)* IEEE, 2016, S. 217–222

[PMG+18] Pichler, Mario ; Meisel, Marcus ; Goranović, Andrija ; Leonhartsberger, Kurt ; Lettner, Georg ; Chasparis, Georgios ; Vallant, Heribert ; Marksteiner, Stefan ; Bieser, Hemma: Decentralized Energy Networks Based on Blockchain: Background, Overview and Concept Discussion. In: *International Conference on Business Information Systems* Springer, 2018, S. 244–257

[PRS07] Pudjianto, Danny ; Ramsay, Charlotte ; Strbac, Goran: Virtual power plant and system integration of distributed energy resources. In: *IET Renewable power generation* 1 (2007), Nr. 1, S. 10–16

[Reg16] Regulation, General Data P.: Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46. In: *Official Journal of the European Union (OJ)* 59 (2016), Nr. 1-88, S. 294

[Rij10] Rijmen, Vincent: Practical-Titled Attack on AES-128 Using Chosen-Text Relations. In: *IACR Cryptology ePrint Archive* 2010 (2010), S. 337

[SDSG+17] Sanseverino, Eleonora R. ; Di Silvestre, Maria L. ; Gallo, Pierluigi ; Zizzo, Gaetano ; Ippolito, Mariano: The Blockchain in Microgrids for Transacting Energy and Attributing Losses. In: *Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2017 IEEE International Conference on* IEEE, 2017, S. 925–930

[SMT11] Saboori, H ; Mohammadi, M ; Taghe, R: Virtual power plant (VPP), definition, concept, components and types. In: *Power and Energy Engineering Conference (APPEEC), 2011 Asia-Pacific* IEEE, 2011, S. 1–4

[Sta01] Standard, NIST-FIPS: Announcing the advanced encryption standard (AES). In: *Federal Information Processing Standards Publication* 197 (2001), S. 1–51

[SW17] Sabounchi, Moein ; Wei, Jin: Towards resilient networked microgrids: Blockchain-enabled peer-to-peer electricity trading mechanism. In: *Energy Internet and Energy*

System Integration (EI2), 2017 IEEE Conference on IEEE, 2017, S. 1–5

[Sza94] Szabo, Nick: Smart contracts. (1994). – `http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html` (accessed: January 8th 2019)

[TJ11] van Tilborg, Henk C. ; Jajodia, Sushil: *Encyclopedia of Cryptography and Security*. Bd. 1. Springer Science & Business Media, 2011

[TNA17] Tanaka, Kenji ; Nagakubo, Kosuke ; Abe, Rikiya: Blockchain-based electricity trading with Digitalgrid router. In: *Consumer Electronics-Taiwan (ICCE-TW), 2017 IEEE International Conference on* IEEE, 2017, S. 201–202

[TS16] Tschorsch, Florian ; Scheuermann, Björn: Bitcoin and beyond: A technical survey on decentralized digital currencies. In: *IEEE Communications Surveys & Tutorials* 18 (2016), Nr. 3, S. 2084–2123

[VD+09] Von Dollen, Don [u. a.]: Report to NIST on the smart grid interoperability standards roadmap. In: *Electric Power Research Institute (EPRI) and National Institute of Standards and Technology* (2009)