Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Prototypische Implementierung von Smart Contracts in einem Blockchain-basierten Smart Grid Szenario der Elektromobilität

Masterarbeit

Studiengang Angewandte Informatik

David Sebastian Seiter

8. März 2021

Erstgutachterin:

Prof. Dr. Christin Schmidt,

Hochschule für Technik und Wirtschaft Berlin

Zweitgutachter:

Dr. Michael Steinhöfel,

Institut für Betriebliche Bildungsforschung

Masterarbeit zur Erlangung des akademischen Grades eines
Master of Science (M.Sc.) im Studiengang
Angewandte Informatik

Prototypische Implementierung von Smart Contracts in einem Blockchain-basierten Smart Grid Szenario der Elektromobilität
Hochschule für Technik und Wirtschaft Berlin
8. März 2021

Vorbemerkung

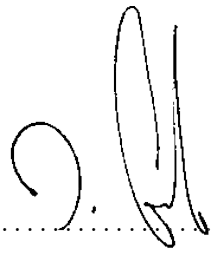
Zur besseren Lesbarkeit wird in der vorliegenden Arbeit das generische Maskulinum verwendet, wobei andere Geschlechteridentitäten mitgemeint sind.

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Berlin, 7. März 2021

Ort, Datum



Unterschrift

Zusammenfassung

In dieser Arbeit wird untersucht, wie energiewirtschaftliche Akteure der Elektromobilität in einem Blockchain-basierten *Smart Energy Grid* agglomeriert werden und über eine *Peer-to-Peer*-Energiehandelsplattform mit *Smart Contracts* Strom dezentral und transparent handeln können. Es wird analysiert, wie damit Fluktuationen in der Stromversorgung beim Einsatz volatiler erneuerbarer Energien ausgeglichen und Bedarfsspitzen bei der zunehmenden Integration von Elektrofahrzeugen mit einer intelligenten Laststeuerung abgefangen werden können. Damit soll die Stromnachfrage flexibilisiert, Sekundärregelleistung ausgeglichen und in letzter Konsequenz Kosten gesenkt und der Stromverbrauch und die -erzeugung aufeinander abgestimmt werden. Dafür werden die notwendigen energietechnischen und Blockchain-technologischen Grundlagen vermittelt, die Anforderungen an die Blockchain-Netzwerkarchitektur analysiert sowie verschiedene Konsensmechanismen, Blockchain-Plattformen und *Smart Contract*-Programmiersprachen einander vergleichend gegenübergestellt. Anschließend erfolgt die prototypische Implementierung verschiedener *Smart Contracts*, um eine *Peer-to-Peer*-Handelsplattform aufzubauen und die Realisierbarkeit des Ansatzes zu zeigen. Im Fokus der Implementierung stehen die Umsetzung der Registrierung, der Zugriffskontrolle und der sicheren Abwicklung von Transaktionen im *Smart Grid* mit Hilfe von *Smart Contracts*. Auf Basis der Implementierung werden Hindernisse der *Ethereum*-Plattform und *Smart Contract*-Sicherheitslücken herausgearbeitet, Problemlösungsstrategien entwickelt und vorgestellt.

Keywords: Blockchain, Smart Contracts, Smart Energy Grid, Smart Contract Design Patterns

This thesis analyses how a *peer-to-peer* energy trading platform can be implemented securely, transparently and without intermediaries using *Smart Contracts* with the objective of compensating for short-term fluctuations in frequency when using volatile renewable energy in the power grid in order to make electricity demand more flexible and reduce costs. For this purpose, the energy and blockchain technology basics are conveyed, the requirements for the blockchain network architecture are analyzed and various consensus mechanisms, blockchain platforms and *Smart Contract* programming languages are compared. Various *Smart Contracts* are implemented based on the analysis to show the feasibility of the approach. In addition, there is a detailed analysis of known *Smart Contract* security loopholes, for which problem-solving strategies are developed and shown.

Inhaltsverzeichnis

1 Einleitung

In diesem Kapitel werden der Hintergrund und die Motivation der Arbeit beleuchtet, die Ziele der Arbeit beschrieben und abschließend der Aufbau der Arbeit dargestellt.

1.1 Motivation

Der Energiesektor befindet sich in einem historischen Transformationsprozess: Mit der Energiewende beschreitet Deutschland den Weg zur Stromversorgung mit erneuerbaren Energien. Durch die zunehmende Integration volatiler erneuerbarer Energien entstehen neben den bisherigen, durch schwankenden Stromverbrauch bedingten, Fluktuationen zusätzliche kurzfristige Schwankungen im Stromnetz, die zur Erhaltung einer stabilen Netzführung ausgeglichen werden müssen. Flexibilität wird zu einem zentralen Paradigma des Energiesektors.

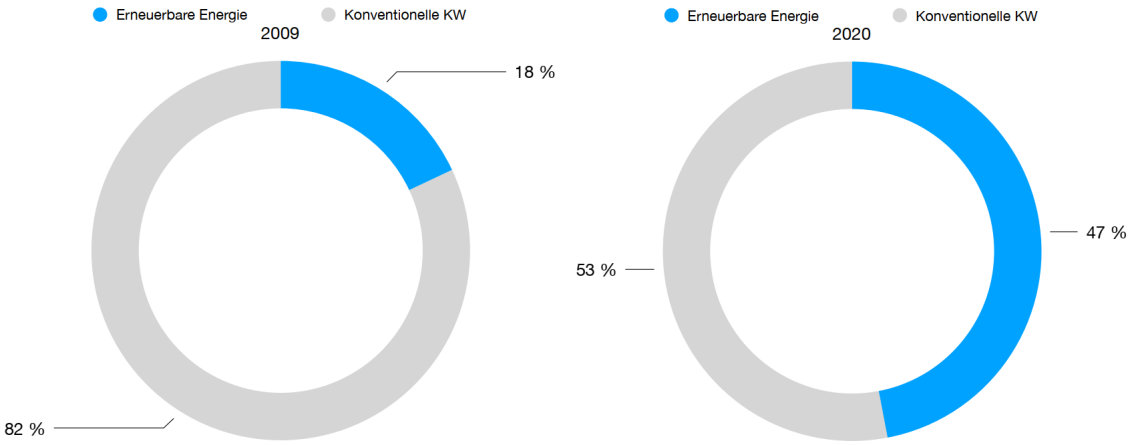


Abbildung 1.1: Nettostromerzeugung in Deutschland 2009 und 2020, eigene Darstellung.

Gleichermaßen befindet sich der Mobilitätssektor im Umbruch: Die Elektromobilität nimmt Fahrt auf. Dabei führt das unkoordinierte Laden einer steigenden Anzahl von Elektrofahrzeugen und der entsprechend aggregierte Strombedarf zu hohen Nachfragespitzen, zusätzlichen Netzengpässen und Überlastungen im Verteilnetz.[13] Um die Volatilität der regenerativen Energieträger ausgleichen und eine intelligente Sektorenkopplung bewältigen zu können, muss die Stromversorgung einen Wandel vollziehen – von der bedarfsgesteuerten Erzeugung hin zum erzeugungsabhängigen Verbrauch in einem intelligenten Stromnetz.[99] Hier können die Speichertechnologien der Elektrofahrzeuge eingesetzt werden, um starke Fluktuationen der Nachfrage abzufangen. Im Bedarfsfall kann mit ihnen überschüssiger

Strom gespeichert und mittels bidirektionalem Energiefluss an das *Smart Grid* abgeben werden, wenn die Nachfrage die Produktion übersteigt. So kann zur Verminderung von Schwankungen der Netzbelastungen beigetragen und die Stabilität und Versorgungssicherheit des Stromnetzes gewährleistet werden. Die Blockchain-Technologie kann für die technische Umsetzung des Energiemanagements eingesetzt werden und die Agglomeration aller Akteure in *Smart Grids* ohne Intermediäre umsetzen. Transaktionen zwischen den verschiedenen Akteuren eines *Smart Grid* können mit der Blockchain dezentral, effizient und sicher realisiert werden. Mit Blockchain-basierten *Smart Grids* kann ein Beitrag dazu geleistet werden, eine intelligente Sektorenkopplung zu gestalten.

1.2 Ziel der Arbeit

Die vorliegende Arbeit analysiert, wie über eine Blockchain-basierte *Peer-to-Peer*-Handelsplattform mit Hilfe von *Smart Contracts* Energie zwischen verschiedenen Akteuren ohne Intermediär gehandelt und so *Smart Energy Grids* der Elektromobilität aufgebaut werden können. Es werden die wichtigen Blockchain-basierten Schlüsseltechnologien erfasst und die modularen Komponenten der Blockchain konsistent und vollständig erläutert sowie wichtige energietechnische Grundlagen dargestellt. Aus diesen Grundlagen werden die Anforderungen zum Aufbau einer dezentralen Blockchain-Netzwerkarchitektur herausgearbeitet und bewertet. Ferner werden unterschiedliche Konsensmechanismen, Blockchain-Plattformen und verschiedene *Smart Contract*-Programmiersprachen unter Aspekten wie des Transaktionsdurchsatzes, der Reaktionsgeschwindigkeit und des Energiebedarfs für die Implementierung des Anwendungsfalls evaluiert. Aufbauend auf den erlangten Erkenntnissen werden Stärken, Schwächen, Chancen und Risiken der Verwendung von *Smart Contracts* skizziert und untersucht, ob damit *Power-to-X*-Maßnahmen, der dezentrale Energiehandel sowie eine intelligente Laststeuerung durch *Demand-Side-Management* im *Smart Grid* umgesetzt und damit regenerative Überschüsse besser genutzt und Schwankungen im Stromnetz automatisiert abgefangen werden können. Im Gesetz zur Digitalisierung der Energiewende stehen außerdem Themen wie die Datenintegrität, der Datenschutz und die Datensicherheit im Fokus.[65] In dieser Arbeit werden daher Gefahren und Sicherheitsschwachstellen in der *Smart Contract*-Programmierung identifiziert, Problemlösungsstrategien dargestellt und untersucht, wie Leistungen von *Smart Contracts* verbessert werden können, um hochfrequenten Stromhandel zu ermöglichen.

1.3 Struktur der Arbeit

Im zweiten Kapitel werden für die kontextuelle Einordnung der Arbeit zunächst die energietechnischen Grundlagen, insbesondere von *Smart Grid*, *Demand-Side-Management* und *Vehicle-to-Grid* erläutert. Anschließend werden die Grundlagen der Blockchain-Technologie

beschrieben und besonders auf die Netzwerkarchitektur, das Protokoll und die Infrastruktur eingegangen. Darauf aufbauend werden kryptographische Grundlagen, Konsensmechanismen und *Smart Contracts* eingeführt. Aus einer Analyse thematisch angrenzender Blockchain-Anwendungen im Energiesektor werden in Kapitel 4 Anforderungen an den Anwendungsfall sowie Stakeholder identifiziert. Im fünften Kapitel erfolgt eine Agglomeration der gewonnen Erkenntnisse. Die einzelnen Komponenten der Blockchain und verschiedene Konsensmechanismen werden analysiert und verschiedene Merkmale des Blockchain-Designs identifiziert, um daraus eine im Kontext des Anwendungsfalls prädestinierte Netzwerkarchitektur abzuleiten. Aus dieser Analyse ergibt sich die Begründung der Plattformentscheidung. Maßgebende Komponenten der *Ethereum*-Plattform sowie die essenziellen Schlüsseltechnologien werden in Kapitel 6 charakterisiert sowie syntaktische Besonderheiten der *Solidity*-Programmiersprache ausgeführt. In Kapitel 7 wird die prototypische Implementierung verschiedener *Smart Contracts* für die Umsetzung eines dezentralen *Peer-to-Peer*-Stromhandels beschrieben. Es werden die Entwicklungsumgebung vorgestellt, maßgebende Bibliotheken abgebildet, Technologieentscheidungen begründet und Handlungsempfehlungen für die Implementierung gegeben. In der prototypischen Implementierung haben sich typische Gefahren der *Smart Contract*-Entwicklung, Einschränkungen der *Ethereum*-Plattform und Limitationen des Blockchain-basierten Ansatzes gezeigt. Im siebten Kapitel werden daher verschiedene Herausforderungen und Schwachstellen diskutiert und konkrete Problemlösungsstrategien und Sicherheitsmechanismen vorgestellt. Abschließend werden Limitationen der Arbeit aufgezeigt, die Ergebnisse dieser Arbeit zusammengefasst und in einen wissenschaftlichen Kontext eingeordnet. Ein Ausblick schließt diese Arbeit ab.

2 Energietechnische Grundlagen

2.1 Einleitung

In diesem Kapitel werden als Einführung in den theoretischen Kontext der Arbeit energietechnische Grundlagen beschrieben. Zunächst werden der Aufbau des Stromnetzes in Deutschland und die dezentrale Stromerzeugung vorgestellt, eine Einführung in das *Smart Grid*, *Demand-Side-Management* und *Vehicle-to-Grid* gegeben sowie deren Bedeutung für den Energiesektor beschrieben.

2.2 Stromnetz in Deutschland

Das Stromnetz bezeichnet in der Energietechnik das Netzwerk zur Übertragung und Verteilung der elektrischen Energie. Es wird in das Verteilnetz und das Übertragungsnetz gegliedert. Mit Hilfe des Übertragungsnetzes wird der Strom von zentralen Großkraftwerken zu den Leistungstransformatoren, die nah an den Verbrauchsschwerpunkten liegen,

weitergeleitet.[73] Das deutsche Übertragungsnetz hat eine Gesamtlänge von ca. 35.000 Kilometern, das Verteilnetz eine Länge von ca. 1,7 Millionen Kilometern.[101]

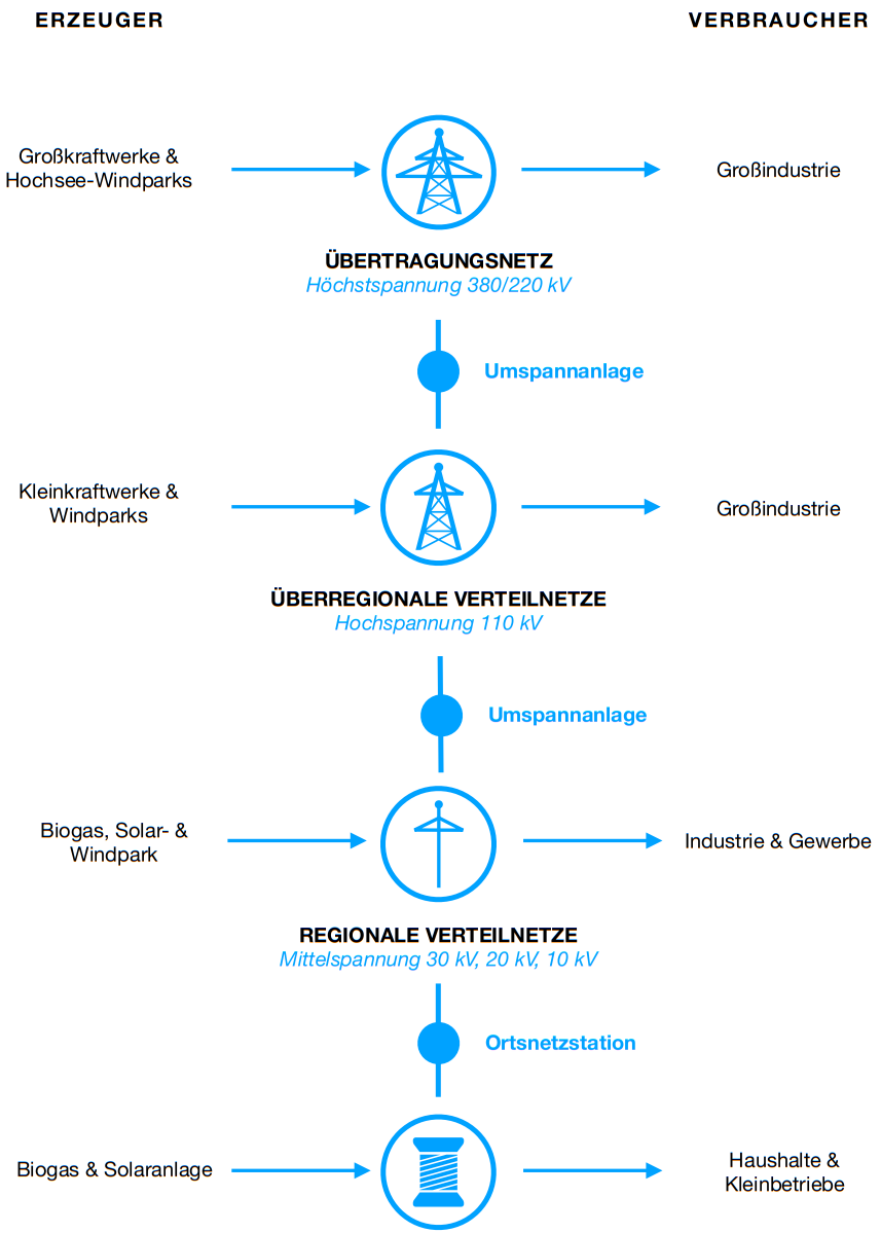


Abbildung 2.1: Übertragungsnetz und Verteilnetz in Deutschland, eigene Darstellung nach [66].

In Deutschland transportieren vier Übertragungsnetzbetreiber - *Tennet TSO*, *Amprion*, *50Hertz Transmission* und *TransnetBW* - den Strom zu rund 900 Verteilungsnetzbetreibern.[72] Die Netzlast betrug im Jahr 2020 474,9 TWh[1].[37] Die Erzeugung aus erneuerba-

[1] 3,2 Prozent unter dem Vorjahreswert

ren Energien betrug 2020 233,1 TWh[2].[37] Da der abgerufene Strombedarf von Faktoren wie Tages- und Jahreszeiten beeinflusst wird, müssen Großkraftwerke auf sich verändernde Anforderungen reagieren und die Stromerzeugung flexibilisieren.[73] Energieversorgungsunternehmen versuchen daher, den Grundlastbedarf möglichst langfristig im Voraus abzuschätzen. Die Grundlast bei der Stromerzeugung bezeichnet den Anteil der elektrischen Leistung in einem Versorgungsgebiet, der im Verlauf eines Tages nicht unterschritten wird. Die Grundlast wird von Grundlastkraftwerken gedeckt, die eine kostengünstige Produktion ermöglichen, jedoch nur schwer zu regeln sind.[73] Wird der Grundlastbedarf überschritten, werden zur Deckung meist zusätzliche Mittel- und Spitzenlastkraftwerke eingesetzt.[68]
Die Spitzenlast bezeichnet den Anteil der elektrischen Leistung, der nur relativ kurzfristig durch Verbrauchsspitzen abgerufen wird, etwa durch gleichzeitiges Laden von Elektrofahrzeugen in einem Versorgungsgebiet am Nachmittag. Zwischen der permanenten Grundlast und den zeitweiligen Spitzenlasten gliedert sich der Bereich der Mittellast ein. Der wesentliche Teil des Mittellast- sowie Spitzenlastbedarfs ist gut vorhersehbar, regelmäßige Verbrauchsspitzen treten zur Mittagszeit sowie am frühen Abend auf.[28] Mit dem Ausbau Erneuerbarer Energien in Deutschland sinkt der Anteil von Kohlekraftwerken an der Stromerzeugung, jedoch kommt es naturgemäß zu höheren Schwankungen im Stromnetz, die durch unstete Lastprofile von Verbrauchern zusätzlich verstärkt werden. Um kurzfristige Spitzenlasten zu kompensieren und Schwankungen im Stromnetz abzuwenden, werden zusätzliche Speicher- und Kraftwerkreserven vorgehalten.[74] Bei dieser Reserve, die für die Stabilisierung des Stromnetzes erforderlich ist, spricht man von Regelleistung oder Regelenergie.[74] Die Grundlage für die Netzstabilität und eine stabile Grundversorgung ist die Deckung des nachgefragten Stroms durch die Energieversorger.[73]

2.3 Smart Grid

Die Volatilität der erneuerbaren Energiequellen sowie die zunehmende Vernetzung einer steigenden Anzahl von Akteuren im Energiesystem bedingt eine steigende Komplexität der Stromversorgung und einen höheren Aufwand für deren Koordination.[100]
Mit *Smart Grids* kann diesen Herausforderungen begegnet werden, indem die Kommunikation, Vernetzung und Steuerung von Erzeugern, Speichern und Verbrauchern im Energieübertragungs- und Verteilnetz sichergestellt wird. Das Ziel ist eine transparente, kosten- und energieeffiziente sowie nachhaltige und dezentrale Energieversorgung, indem die Erzeugung, Speicherung und der Verbrauch des Stroms aufeinander abgestimmt werden, um Schwankungen auszugleichen. In *Smart Grids* werden hierfür Informations- und Kommunikationstechnologien sowie Energiemanagementsysteme verwendet, um Stromerzeuger, Verteilstationen und Verbraucher miteinander zu verbinden.

[2] 4,1 Prozent über dem Vorjahreswert

In einem *Smart Grid* können viele Akteure in das Stromnetz eingegliedert werden und es ermöglicht ihnen, in einem kleinen, individuellen Maßstab, Strom an das Stromnetz abzugeben.

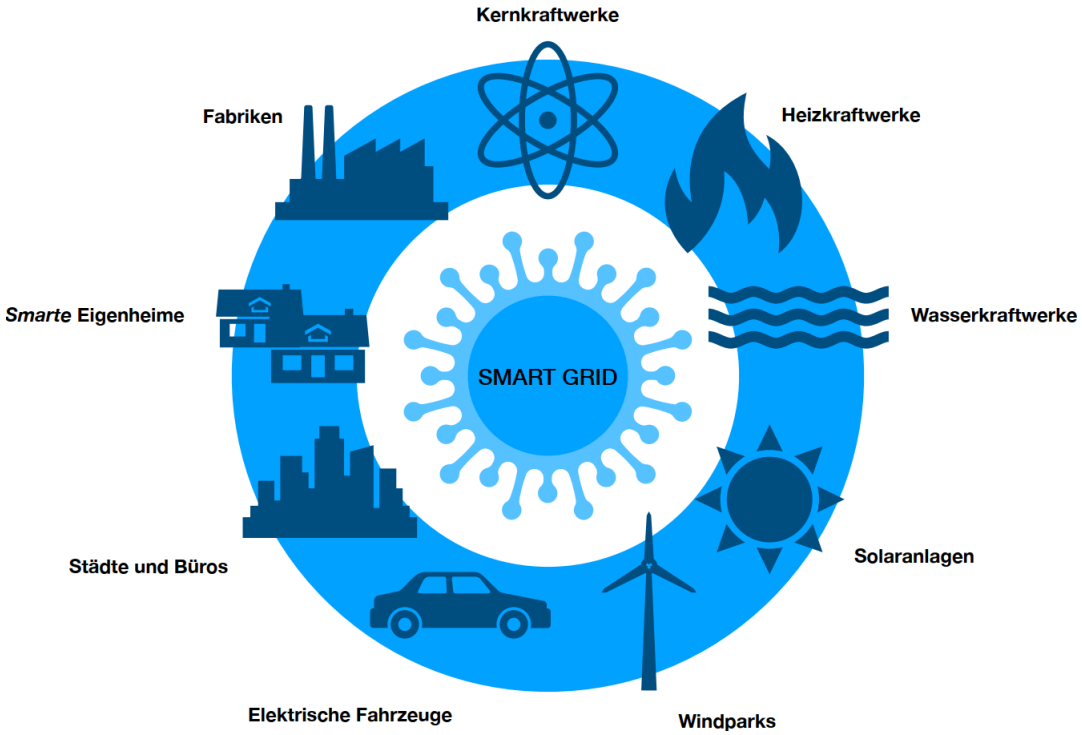


Abbildung 2.2: Unterschiedliche Akteure bilden ein intelligentes Netzwerk: Das *Smart Grid*, eigene Darstellung.

Gleichzeitig erhöht sich jedoch die Komplexität des Energiesystems. Da in Echtzeit Informationen zum aktuellen Verbrauch und zur Energieproduktion verarbeitet werden, wird die Stromverteilung dezentral, effizient und flexibel umgesetzt. Durch ein intelligentes Steuerungsmanagement wird eine optimale Auslastung des Stromnetzes angestrebt, die die Netzstabilität aufrechterhält und die sich aktuellen Gegebenheiten flexibel anpasst. Damit diese Herausforderung geleistet werden kann, müssen die Komponenten eines *Smart Grid* miteinander vernetzt sein. So können Zustandsinformationen oder Lastflussdaten kommuniziert werden.[73] Auf Verbraucherseite werden intelligente Messsysteme[3] installiert, die den aktuellen Stromverbrauch weitergeben.

[3] Beispielsweise ein *Smart Meter* oder eine *Wallbox* mit entsprechendem *Smart Meter Gateway*

2.4 Vehicle-to-Grid

Damit Elektrofahrzeuge zur Verbesserung des Netzbetriebs sowohl bei Bedarf Stromspitzen aufnehmen, als auch bei erhöhter Nachfrage wieder ins Stromnetz einspeisen können, bedarf es bidirektional ladefähiger Elektrofahrzeuge. Mit *Vehicle-to-Grid* können die Energiespeicher der Fahrzeuge als Puffer genutzt und Energie zu Zeiten erhöhten Energiebedarfs wieder an das Stromnetz abgegeben werden.

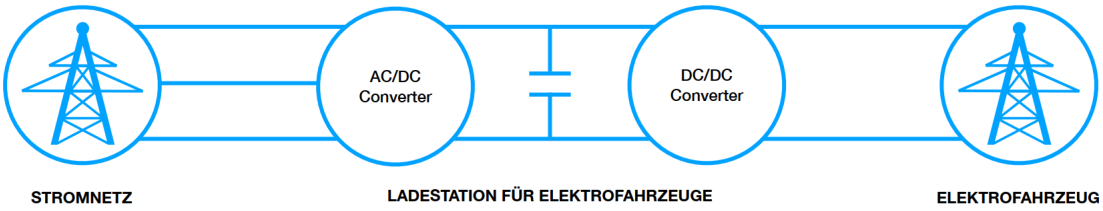


Abbildung 2.3: Leistungsflussdiagramm für *Vehicle-to-Grid*, eigene Darstellung nach [109].

Vehicle-to-Grid beschreibt den Energieaustausch zwischen Elektrofahrzeugen und dem Stromnetz. Dafür müssen in den Elektrofahrzeugen *AC/DC-Wandler*[4] und *DC/DC-Wandler* verbaut sein. Der *AC/DC-Wandler* wird verwendet, um den Wechselstrom vom Stromnetz in Gleichstrom beim Laden des Elektrofahrzeugs und umgekehrt beim Entladen umzuwandeln. Der *DC/DC-Wandler* ist für die Steuerung des bidirektionalen Leistungsflusses verantwortlich und fungiert während des Ladens bzw. Entladens als Abwärts- oder Aufwärtswandler.[105]

2.5 Demand-Side-Management

Die Flexibilität des Stromnetzes wurde bis dato von konventionellen Kraftwerken und Speichern gewährleistet. Mit dem Wandel von der bedarfsgesteuerten Erzeugung hin zum erzeugungsabhängigen Verbrauch steigt auch die Relevanz der Nachfrageflexibilität.[99] Laststeuerung durch *Demand-Side-Management* bezeichnet die Steuerung der Nachfrage bei Abnehmern in Industrie, Gewerbe und Privathaushalten. Man unterscheidet zwischen implizitem, anreizbasiertem und explizitem sowie preisbasiertem *Demand-Side-Management*.[102] Prozesse des *Demand Side Managements* können mit *Smart Contracts* in Echtzeit gesteuert und optimiert werden. Transaktionen werden im Anschluss in die Blockchain geschrieben und sind so sicher und dezentral gespeichert.

[4] AC für Wechselstrom und DC für Gleichstrom

2.6 Fazit

In diesem Kapitel wurde mit der Beschreibung der energietechnischen Grundlagen in den theoretischen Kontext der Arbeit eingeführt. Die zunehmende Integration volatiler erneuerbarer Energien und das Aufkommen zunehmend mehr dezentraler Energieerzeugungsanlagen führen zu höheren Schwankungen und steigern den Koordinierungsaufwand des Stromnetzes.[96] Der steigenden Komplexität des Stromnetzes kann dabei mit einem hohen Automatisierungsgrad und Flexibilisierungsmaßnahmen begegnet werden. Flexibilisierungsmaßnahmen, wie intelligentes *Demand-Side-Management*, *Demand-Response-Management*, Elektrofahrzeuge als Stromspeicher sowie die Integration von *Power-to-X*-Maßnahmen können gleichzeitig zur Netzstabilität in *Smart Grids* beitragen.[108] Dadurch können mehr Anlagen in das Netz integriert, der Abruf von Regelenergie reduziert und in letzter Instanz durch intelligentes Lastmanagement erneuerbare Energien besser integriert werden. Dieses Kapitel hat bereits erste Herausforderungen gezeigt. In einem *Smart Grid* der Elektromobilität entstehen durch den Datenaustausch und die Bereitstellung von Echtzeitinformationen enorme Datenmengen. Außerdem herrschen im Energiesektor hohe Sicherheitsstandards und hohe Ansprüche an die Performanz des Energiesystems. Daher bedarf es einer Technologie, die den Stromhandel effizient und sicher umsetzt und Konsens zwischen allen Akteuren aushandelt. Damit es zu keiner Überlastung kommt sowie die Grenzwerte der Netzspannung eingehalten werden können, muss ein nachvollziehbares, transparentes und sicheres Kommunikations- und Handelssystem geschaffen werden. Die Blockchain-Technologie hat das Potenzial dies zu leisten und die zunehmende Vernetzung der im *Smart Grid* beteiligten, autonomen Geräte mit dem Einsatz von *Smart Contracts* sicherzustellen.[6]

3 Blockchain-Technologische Grundlagen

3.1 Einleitung

In diesem Kapitel werden die Grundlagen der Blockchain beschrieben. Zunächst werden mit der Netzwerkarchitektur, der Infrastruktur und dem Blockchain-Protokoll die technologischen Grundlagen skizziert. Für ein gutes Verständnis der Funktionsweise der Blockchain werden im Anschluss die kryptografischen Grundlagen erläutert. Anschließend werden verschiedene Konsensmechanismen beschrieben und auf ihre Eignung für den Anwendungsfall untersucht. Das Kapitel schließt mit einer Einführung in *Smart Contracts* ab.

3.2 Blockchain

Das Konzept der Blockchain als verteiltes Datenbankmanagementsystem wurde erstmals 2008 unter dem Pseudonym *Satoshi Nakamoto* im White-Paper *Bitcoin: A peer-to-peer Electronic Cash System* vorgestellt.[81] Ein Jahr später wurde unter dem selben Pseudonym eine erste Referenzimplementierung der Kryptowährung *Bitcoin*[5] veröffentlicht - und damit die erste öffentliche, verteilte Blockchain begründet.[82]
Die Blockchain ist eine spezielle Form der *Distributed-Ledger*-Technologie, bei der im Gegensatz zu konventionellen *Centralized-Ledger*-Technologien alle Transaktionsinformationen dezentral bei einer Vielzahl der teilnehmenden Netzwerkakteure gespeichert werden.

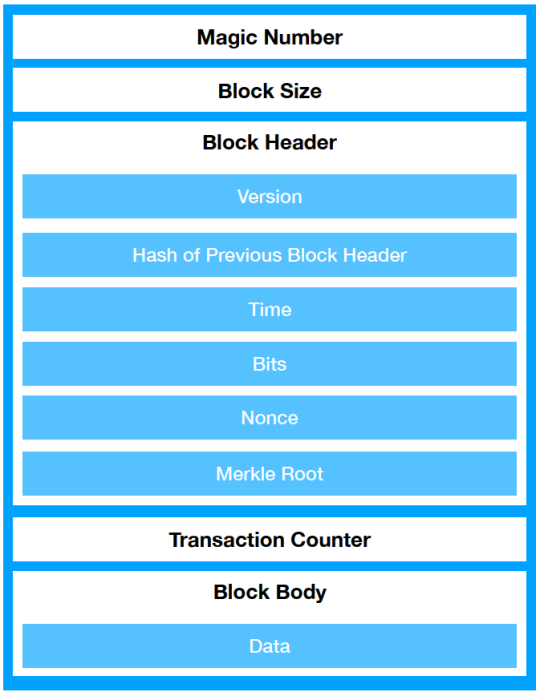


Abbildung 3.1: Aufbau eines Blockes der Blockchain, eigene Darstellung nach [53].

Zu den entscheidenden Eigenschaften der Blockchain gehört, dass Daten und Transaktionsdetails, die im gemeinsamen *Ledger* aufgezeichnet wurden, hinterher nicht mehr verändert werden können (*immutability*). Sichergestellt wird das über die Verknüpfung aller Blöcke - jeder Informationsblock generiert seperat einen kryptografischen *Hashwert* aus einer alphanumerischen Zeichenfolge. Jeder Block enthält dabei nicht nur die eigene digitale Signatur, sondern auch die des vorherigen Blockes. Transaktionen können lediglich korrigiert werden, indem neue, ausgleichende Transaktionen hinzugefügt werden. Die Informationen

[5] https://bitcoin.org/

oder Transaktionsdetails, die in das gemeinsame *Ledger* aufgenommen wurden, können von allen Akteuren des Netzwerks eingesehen werden. Dies stellt sicher, dass alle Akteure die gleichen Informationen teilen und darauf vertrauen können, dass die der Blockchain hinzugefügten Daten valide sind. Durch die öffentliche Einsicht, Nichtveränderbarkeit und Kontrollierbarkeit der Informationen entsteht Vertrauen ohne eine zentrale Schnittstelle (*trustlessness*).[112]

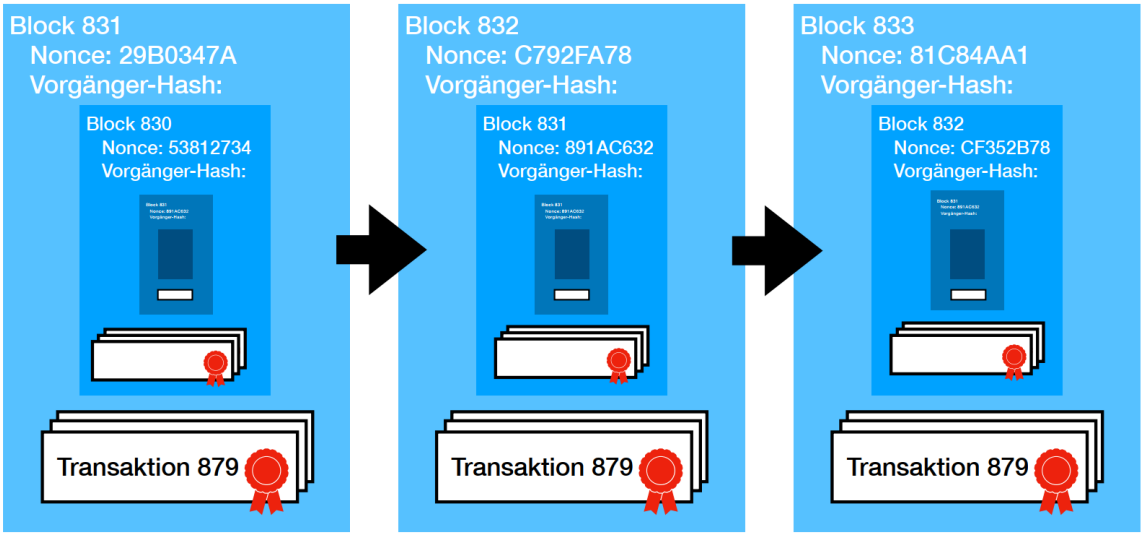


Abbildung 3.2: Verknüpfung von Blöcken in der Blockchain, eigene Darstellung.

Das Entfallen der Abhängigkeit von einer zentralen Instanz und die Verteilung der Transaktionsinformationen auf allen *Nodes* löst vielschichtige Probleme der Datenintegrität von *Centralized-Ledger*-Technologien und beugt Datenmanipulationen vor. Der zentrale Bestandteil der Blockchain, der Block, ist Träger der Daten und Transaktionsdetails und besteht aus verschiedenen Komponenten. Die Blöcke bilden bei mehreren Transaktionen eine Blockkette (engl.: Blockchain). Die Transaktionen und Blöcke der Blockchain werden aufeinander aufbauend gespeichert, da jedem *Block-Hash* bei der Generierung ein Zeitstempel zugefügt wird. Sobald sich ein Wert in einem Block ändert, müssen alle anderen Werte in der Blockchain entsprechend angepasst werden, damit die Integrität der gesamten Blockchain nicht beeinträchtigt wird. In der Praxis benötigt diese Anpassung eine hohe Rechenleistung und Zeit, weshalb das Ändern oder Löschen eines Wertes kaum möglich ist.

3.2.1 Netzwerkarchitektur

Im Gegensatz zu usuellen Datenbanken in Form einer zentral organisierten *Client-Server*-Architektur besteht die Blockchain aus einer dezentral verteilten Datenbankstruktur mit mehreren *Nodes*, von denen jeder jeweils eine vollständige Kopie der Blockchain besitzt. Bei zentral organisierten Datenbanken wird jede Interaktion über eine zentrale Instanz abgewickelt, bei dezentral verteilten Datenbankstrukturen ist keine zentrale Instanz vorhanden, sodass die Interaktion direkt zwischen den Netzwerkteilnehmern verläuft. Dabei gibt es keine Hierarchien untereinander. Der Aufbau entspricht dem einer klassischen *Peer-to-Peer*-Netzwerkarchitektur.

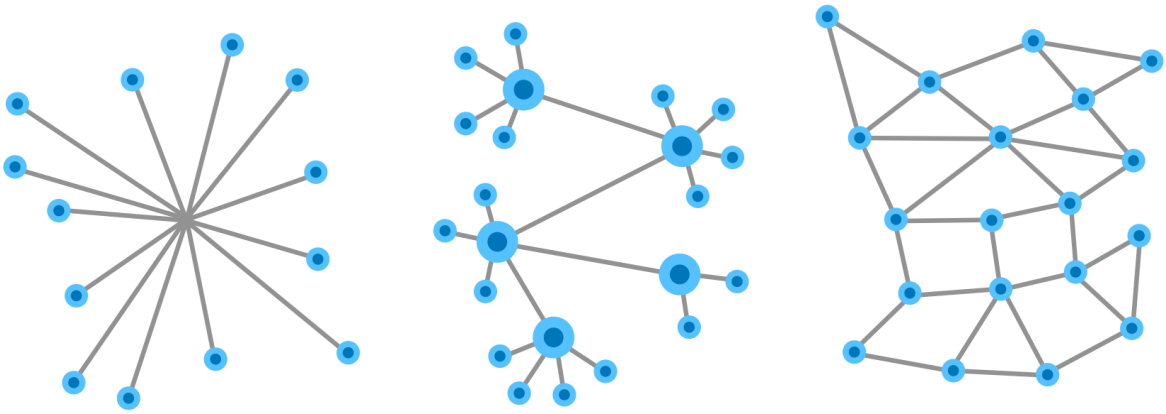


Abbildung 3.3: Struktur eines zentralen, dezentralen und verteilten Netzwerks, eigene Darstellung.

3.2.2 Protokoll

In der Blockchain werden alle Arbeitsabläufe und deren Reihenfolge, wie die Algorithmen zur Steuerung und Validierung von Transaktionen oder die Mechanismen für die Kommunikation der Akteure, von allen *Nodes* im Netzwerk in einem vordefinierten Protokoll festgehalten. Das Protokoll umfasst das gesamte Regelwerk, nach dem das Netzwerk aufgebaut ist.[2] In diesem wird bei der Entwicklung der Blockchain definiert, welcher *Hashing*-Algorithmus verwendet wird, welchen Schwierigkeitsgrad der Validierungsprozess hat, wie die Vergütung der Validierung (engl.: *rewarding*) abläuft und wie die Validität über den Konsensmechanismus durch das Netzwerk bestätigt wird. Wenn sich die Regeln der Blockchain ändern, beispielsweise durch Anpassungen im Code oder des Protokolls, spaltet sich diese in parallel verlaufende Pfade auf. Bei diesem Prozess wird von einem *Fork* gespro-

chen, die koexistierende Blockchain wird als *Side Fork* bezeichnet.[84] Dabei wird zwischen einem *Soft Fork* und einem *Hard Fork* unterschieden:

Soft Fork *Soft Forks* bieten Abwärtskompatibilität, sodass Netzwerkteilnehmer trotz *Fork* gemeinsam an einer Blockchain arbeiten. Die validierten Blöcke bestehen bei einem *Soft Fork* nur temporär in einem *Side Fork* und werden nach einer gewissen Zeit wieder der Blockchain hinzugefügt.[84]

Hard Fork *Hard Forks* entstehen durch Änderungen des Konsensmechanismus oder des Protokolls der Blockchain. Sie bieten keine Abwärtskompatibilität. Da die Forks unterschiedlichen Protokollen mit verschiedenen Regeln unterliegen, entstehen unterschiedliche Blöcke, die nicht mehr miteinander kompatibel sind.

Um das Blockchain-Protokoll, den Konsensmechanismus oder den Validierungsprozess nachträglich dennoch ändern zu können, kann auf *Sidechains* zurückgegriffen werden. Eine *Sidechain* ist die Abzweigung einer übergeordneten Blockchain (engl.: *Mainchain*). Die *Sidechain* bleibt aber lose mit der *Mainchain* in Verbindung und erlaubt so beispielsweise das Übertragen von *Token*. Somit ist auch eine Kommunikation zwischen verschiedenen Blockchains realisierbar. Dennoch sind *Sidechain* und *Mainchain* für den Verlauf der Abspaltung voneinander unabhängig, können jedoch im Gegensatz zum *Fork* zu einem späteren Zeitpunkt wieder vereint werden.[84] *Sidechains* bieten die Möglichkeit, alternative Funktionalitäten zu implementieren. Daneben eignet sich die *Sidechain* auch als Mittel zum Testen von Änderungen und Neuerungen an der *Mainchain*, da sie isoliert von ihr arbeitet.

3.2.3 Infrastruktur

Mit der Infrastruktur werden die Möglichkeiten der Teilnahme am Netzwerk und damit die Zugriffsrechte auf Netzwerkdaten definiert. Man unterscheidet zwischen drei verschiedenen Ausprägungsarten. Im ersten Fall, der *Public Blockchain*, herrschen keinerlei Restriktionen hinsichtlich der Teilnahme an Validierungsprozessen und der Konsensbildung, was eine erhöhte Ausfallsicherheit durch die entstehende hohe Redundanz von Knoten und Validatoren im *Smart Grid* bewirkt, jedoch auch Veränderungen am Protokoll der Blockchain auf Grund des herrschenden Mehrheitsprinzips erschwert. Im Gegensatz zu *Public Blockchains* ist bei *Private Blockchains* die Teilnahme am Netzwerk nur bestimmten Akteuren erlaubt und es gibt eine zentrale Autorität, die die Teilnahme reguliert. Nur die zentrale Autorität kann bestimmte Rechte, wie beispielsweise das Initiieren von Transaktionen, vergeben. Eine Sonderform der *Private Blockchain* stellt die *Consortium Blockchain* dar, bei der die

Validierung von Transaktionen auf ein Konsortium verteilt wird, welches aus mehreren Teilnehmern besteht. Typischerweise erfolgt die Konsensfindung dabei durch die Mehrheit der berechtigten Teilnehmer.[2] *Hybrid Blockchains* vereinen Eigenschaften der *Public* und der *Private* Blockchain. So sind die administrativen Rechte zum Einsehen, Modifizieren und Genehmigen von Transaktionen auf ausgewählte Teilnehmer beschränkt. Eine Transaktion in einer *Hybrid Blockchain* wird zunächst im privaten Zustand abgewickelt und von den ausgewählten Teilnehmern validiert. Anschließend wird sie im öffentlichen Zustand der Blockchain als ein unveränderlicher Datensatz gespeichert.

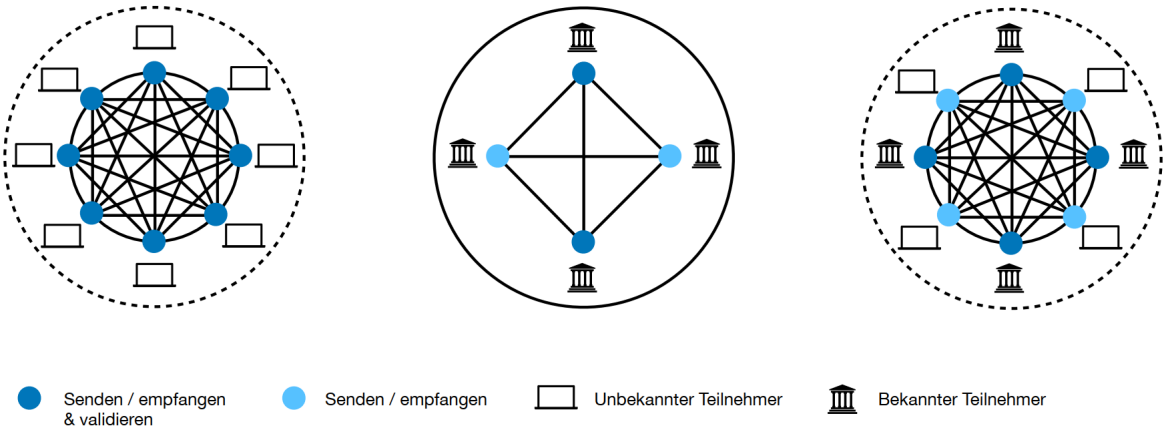


Abbildung 3.4: Unterschiedliche Typen der Blockchain-Technologie, eigene Darstellung nach [90].

Die Infrastrukturen der Blockchain bestimmen auch die Teilnahme am Validierungsprozess. Es wird zwischen der uneingeschränkten (*permissionless*) und der eingeschränkten Teilnahme (*permissioned*) unterschieden. Wenn Blockchains mit Systemen außerhalb des Blockchain-Netzwerks kommunizieren, *Assets* und Informationen übertragen und die Konsistenz zwischen den unterschiedlichen Systemen gewährleistet ist, wird von Intraoperabilität (*Intraoperability*) gesprochen.[107]

3.3 Kryptographische Grundlagen

Kryptographische Verschlüsselungsmethoden gehören zu den wichtigsten Grundlagen der Blockchain-Technologie im Kontext des *Smart Grid*. Sie werden für das *Mining* von Blöcken, die Authentifizierung der Teilnehmer, das Überprüfen der Authentizität von Nachrichten oder das Erzeugen von Konsens angewandt. Die meisten Blockchains verwenden erprobte kryptografische Primitive, wie beispielsweise Hashfunktionen, asymmetrische Kryp-

tographie oder kryptografisch sichere Zufallsgeneratoren, die nachfolgend beschrieben werden.

Symmetrische Kryptographie Bei der symmetrischen Kryptographie wird derselbe Verschlüsselungsalgorithmus (*Cipher*) zur Ver- und Entschlüsselung der Nachricht verwendet. Zwei Parteien nutzen hierfür einen gemeinsamen *Private Key*. Offensichtlicher Nachteil der symmetrischen Kryptographie ist, dass der *Cipher*-Algorithmus geheim gehalten und der Schlüsselaustausch über einen sicheren Kanal erfolgen muss. Weiterer Nachteil ist die Authentizität der Nachrichten: Es muss belegt werden, dass eine Partei eine Nachricht versendet hat. Beiden Problemen bei der symmetrischen Kryptographie kann durch asymmetrische Kryptographie[6] begegnet werden.

Asymmetrische Kryptographie Bei der asymmetrischen Kryptographie werden unterschiedliche *Cipher* zum Ver- und Entschlüsseln - ein *Public Key* und ein *Private Key* - verwendet. Der *Public Key* ist dabei jedem bekannt, der *Private Key* muss geheim gehalten werden. *Private* und *Public Key* müssen dabei als Schlüsselpaar angewendet werden - der *Ciphertext* kann nicht mit demselben Schlüssel entschlüsselt werden, mit dem er zuvor verschlüsselt wurde.

Abbildung 3.5: Funktionsweise der asymmetrischen Verschlüsselung, eigene Darstellung nach [2].

Private und *Public Key* sind mathematisch miteinander verwandt - der *Public Key* wird aus dem *Private Key* generiert. Der durch das *Public-Key*-Verschlüsselungverfahren generierte *Ciphertext* ist daher sicherer, als der der symmetrischen Kryptographie. Das Schlüsselpaar

[6] *Public-Key*-Verschlüsselung

wird mit Operationen der *Elliptic Curve Cryptography* generiert. Sowohl für die Authentifizierung von Netzwerkteilnehmern als auch für die Autorisierung der Transaktionen wird das *Public-Key*-Verschlüsselungverfahren verwendet.

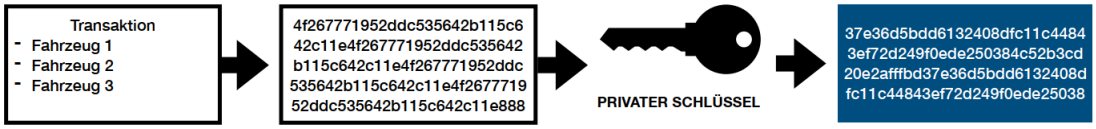


Abbildung 3.6: Schematische Erstellung der digitalen Signatur, eigene Darstellung nach [2].

Für die Benutzerauthentifizierung wird für jeden Nutzer ein zufällig generierter *Private Key* erstellt. Der zugehörige *Public Key* wird anschließend mathematisch errechnet. Der generierte *Public Key* dient als eindeutig identifizierbare öffentliche Adresse des Teilnehmers, an die nun Transaktionen versendet werden können. Der *Private Key* dient zur Authentifizierung beim Zugriff auf diese Adresse, mit dem die Transaktionen gelesen werden können. Für die Autorisierung von Transaktionen in der Blockchain kommen zusätzlich digitale Signaturen zum Einsatz. Neben der asymmetrischen Kryptographie verwenden digitale Signaturen Hashfunktionen.

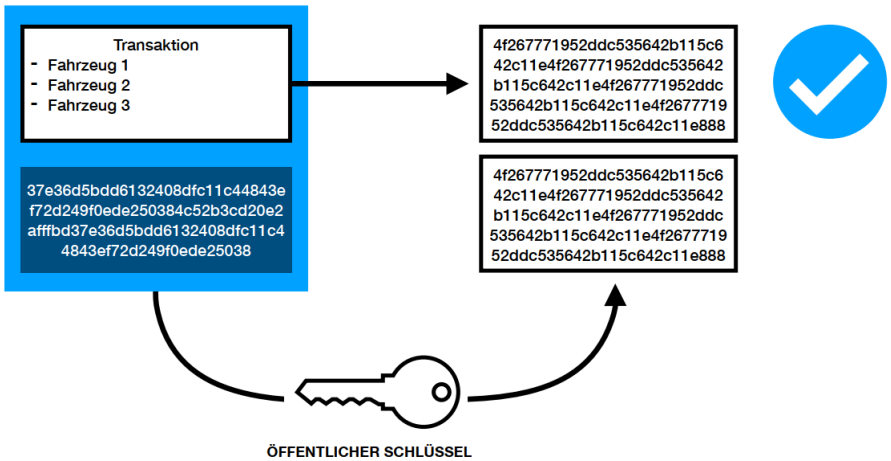


Abbildung 3.7: Überprüfung einer Nachricht anhand der digitalen Signatur, eigene Darstellung nach [2].

Um eine Transaktion digital zu signieren, übergibt der Kontoeigentümer, von dessen Konto

die Transaktion ausgehen soll, zunächst die in der Transaktion enthaltenen Informationen an eine Hashfunktion. Der daraus entstehende Hashwert wird im Anschluss mit dem privaten Schlüssel des Kontoeigentümers verschlüsselt. Die in der Transaktion enthaltenen Informationen werden mit der digitalen Signatur in einer Datei kombiniert und an alle Teilnehmer des Blockchain-Netzwerks gesendet. Der Empfänger der Datei berechnet erst den Hashwert der Information. Danach kann er mit dem öffentlichen Schlüssel des Kontoeigentümers die digitale Signatur entschlüsseln. Wenn er als Ergebnis den Hashwert erhält, der dem Hashwert der Information entspricht, ist sichergestellt, dass die Transaktion nicht manipuliert wurde. Demnach dienen digitale Signaturen als Nachweis, dass eine Transaktion von einem bestimmten Kontoeigentümer stammt und dieser der Transaktion zweifelsfrei zugestimmt hat. Zudem wird sichergestellt, dass die Transaktion nachträglich nicht verändert wurde.

Hashfunktionen Um die große Anzahl an Daten und Informationen schnell und eindeutig referenzieren zu können, werden bei der Blockchain-Technologie auf die Informationen Hashfunktionen angewendet.[2]

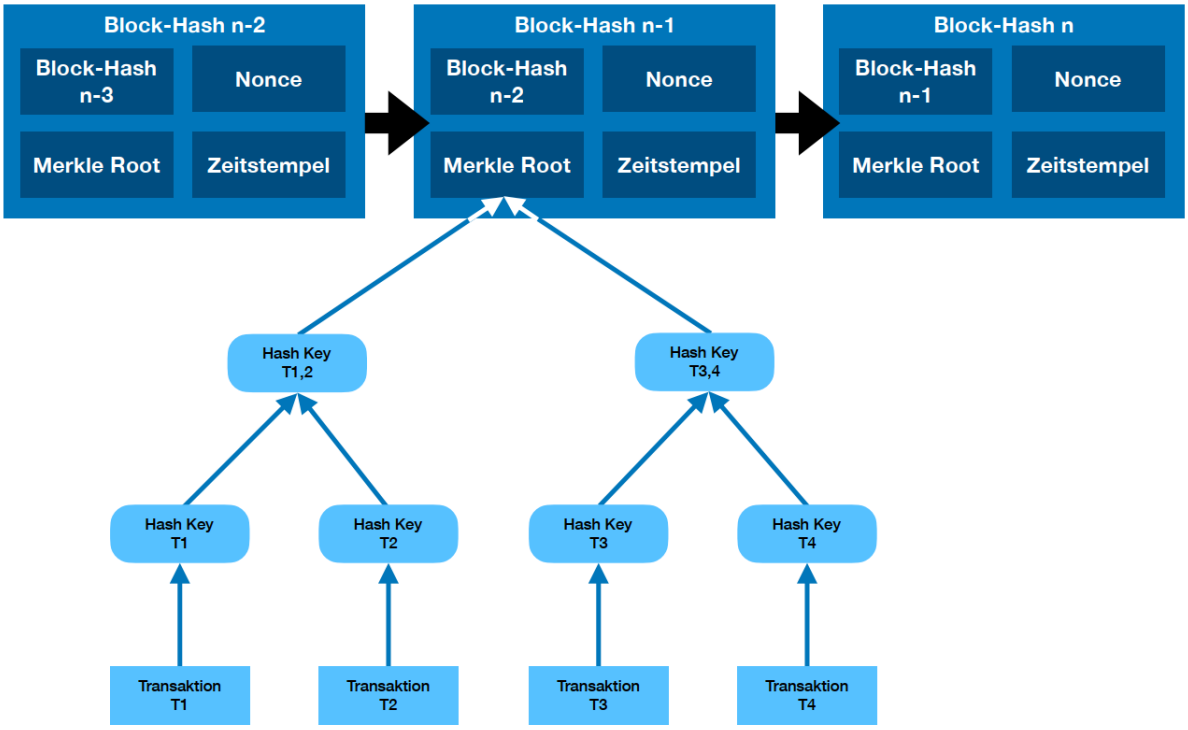
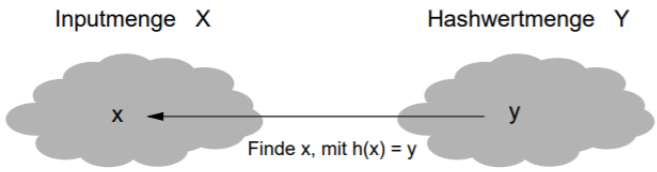


Abbildung 3.8: Verkettung der Blöcke in einer Blockchain, eigene Darstellung nach [16].

Sie berechnen aus den unterschiedlich langen Informationen Bitfolgen mit fester Länge. Aus den Daten wird so ein einzigartiger Hashwert erstellt, mit dem sie eindeutig identifizierbar sind. An die Hashfunktionen gibt es drei grundlegende kryptographische Anforderungen: Die Urbildresistenz, die schwache sowie die starke Kollisionsresistenz.[46] Nach dem Prinzip der Urbildresistenz darf es rechnerisch nicht möglich sein, aus dem erzeugten Hashwert die ursprüngliche Eingangsinformation zu berechnen.

$$geg.: h: X \rightarrow Y, \quad Hashwert \quad y \in Y \tag{1}$$

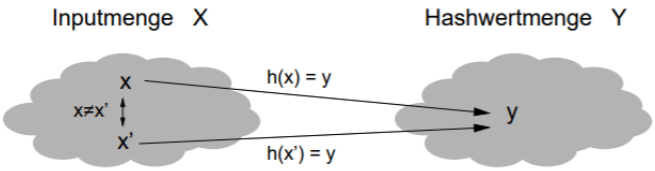
$$ges.: x \in X, \quad mit \quad h(x) = y \tag{2}$$



Die schwache Kollisionsresistenz besagt, dass für eine gegebene Eingangsinformation und den dazugehörigen Hashwert keine zweite Information vorhanden sein darf, die denselben Hashwert besitzt.[46]

$$geg.: h: X \rightarrow Y, \quad Text \quad x \in X \tag{3}$$

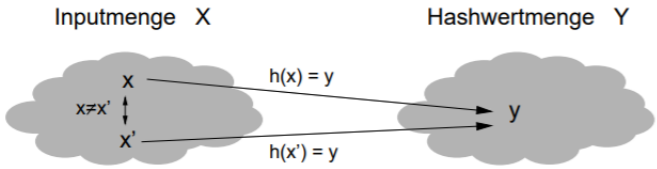
$$ges.: x' \in X, \quad mit \quad x' \neq x \quad und \quad h(x') = h(x) \tag{4}$$



Nach der starken Kollisionsresistenz darf es keine zwei verschiedenen Eingangsinformationen geben, die denselben Hashwert besitzen.[46]

$$geg.: h : X \to Y \tag{5}$$

$$ges.: x, x' \in X, \quad mit \quad x' \neq x \quad und \quad h(x') = h(x) \tag{6}$$



Mit der Hashfunktion werden sowohl die Transaktionen, als auch jeder Block zu Hashwerten umgewandelt. Zunächst wird von jeder Transaktion eines Blocks der jeweilige Hashwert berechnet. Diese werden anschließend paarweise gebündelt und nochmals gehasht, bis alle Transaktionen zu einem einzigen Hashwert gebündelt sind. Das letzte Bündel bildet den *Root Hash* oder *Merkle Root*.[16] Die beschriebene Struktur der paarweisen Bündelung der Hashwerte bildet den *Merkle Tree*.

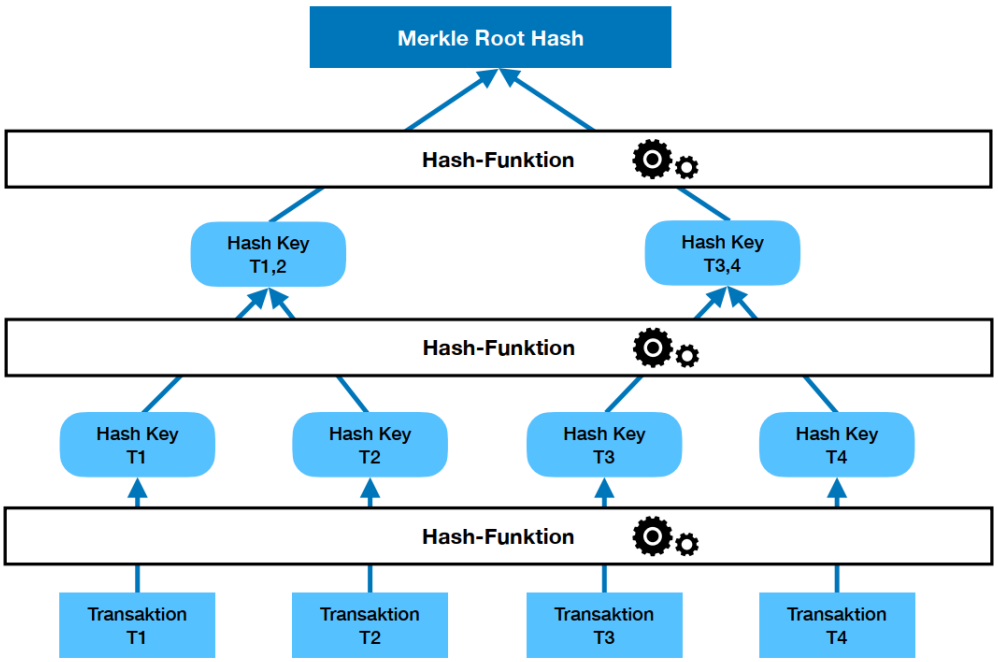


Abbildung 3.9: Struktur eines *Merkle Trees* der Blockchain, eigene Darstellung nach [16].

Mit dem *Merkle Tree* können Manipulationen in der Blockchain schnell erkannt werden: ist eine Transaktion manipuliert, ändern sich die Hashwerte bis zum Hashwert des *Merkle Root Hash*. Wenn die Hashwerte nicht mehr mit den mittels Konsensmechanismus validierten Werten in der *Mainchain* übereinstimmen, wird die manipulierte Kette vom Netzwerk ignoriert.[16] Um Speicherengpässe zu vermeiden, werden nicht alle Hashwerte der Transaktionen, sondern lediglich der *Merkle Root* in dem *Hash* des Blocks[7], gespeichert. Der *Block-Hash* besteht zusätzlich aus einem Zeitstempel, dem Hashwert des Vorgängerblocks sowie der *Nonce*, einer zufälligen Zahl.[16]

3.4 Konsensmechanismen

Der Konsensmechanismus (engl.: *consensus mechanism* oder *consensus protocol*) kann als die grundlegende Innovation sowie zentrale und kritische Funktion hinter der Blockchain-Technologie betrachtet werden. Der Algorithmus bestimmt die Regeln und Mechanismen, wie neue Blöcke validiert werden und die Beteiligten sich auf eine Transaktionshistorie einigen. So wird die Authentizität der darin enthaltenen Transaktionen gewährleistet. Der Konsensmechanismus variiert zwischen verschiedenen Blockchain-Technologien.[76] Jeder Konsensmechanismus bringt Vor- und Nachteile in Bezug auf die Transaktionsgeschwindigkeit, die Energieeffizienz, die Skalierbarkeit oder der Manipulationssicherheit mit sich, die für verschiedene Anwendungsfälle bewertet werden müssen.[76] Die Regeln des Konsensmechanismus bilden den Rahmen des Validierungsprozesses. Mit *Bitcoin* wurde der *Proof-of-Work*-Algorithmus eingeführt.[81]. Seither haben sich weitere, alternative Konsensmechanismen hervorgetan, wie etwa der *Proof-of-Stake*, *Proof-of-Benefit* oder der *Practical Byzantine Fault Tolerance*-Konsensmechanismus. Im Folgenden werden verschiedene Konsensmechanismen vorgestellt, die vor allem auf Grund der Faktoren Performanz und Sicherheit für den Anwendungsfall in Frage kommen.

Proof-of-Work Der Grundgedanke des *Proof-of-Work*-Konsensmechanismus ist, dass an der Konsensbildung beteiligte *Miner* eine kryptografische Rechenaufgabe lösen müssen, um Informationen in einen neuen Block zu schreiben und diesen der Blockchain hinzufügen zu dürfen. Für das Lösen der kryptografischen Rechenaufgabe, dem *Mining*, wird der *Hashcash*-Algorithmus verwendet, welcher sich auf *Hash*-Funktionen, wie *SHA-256*, stützt und zur Verhinderung von *Denial-of-Service*-Attacken entwickelt wurde.[33] Die *Miner*, die als erstes die valide Lösung der kryptografischen Aufgabe errechnen, dürfen den nächsten Block an die Blockchain anhängen und erhalten als Kompensation eine monetäre Gegenleistung. Hierfür müssen die Miner mittels *Hash*-Algorithmus einen neuen Hashwert bestimmter Länge erzeugen. Diesen erzeugen sie aus anstehenden Transaktionen und dem *Block-Hash*

[7] auch *Block-Hash* oder *Block-Header* genannt

des vorherigen Blocks. Dabei kann die erforderliche Rechenkapazität durch die Schwierigkeit der Berechnung beeinflusst werden. Da es keine Methode gibt, mit der der Hashwert berechnet werden kann, können Miner nur durch Ausprobieren herausfinden, wann das Ergebnis den geforderten Kriterien des Zielwertes entspricht. Miner nutzen hierfür eine *nonce*, eine zufällig gewählte Zahl, die in die Berechnung des Hashwertes mit einfließt. Die *nonce* wird so lange iteriert, bis das Ergebnis des Hashwertes unter dem entsprechenden Zielwert liegt. Sobald der Hashwert den Zielwert unterschreitet, hat der Miner die Aufgabe gelöst und der Block wird der Blockchain hinzugefügt. Dieser Algorithmus eignet sich sehr gut für anonyme, offene Netzwerke, in denen der Wettbewerb um Kryptowährung die Sicherheit im Netzwerk fördert, nicht jedoch für private oder Konsortialnetzwerke.

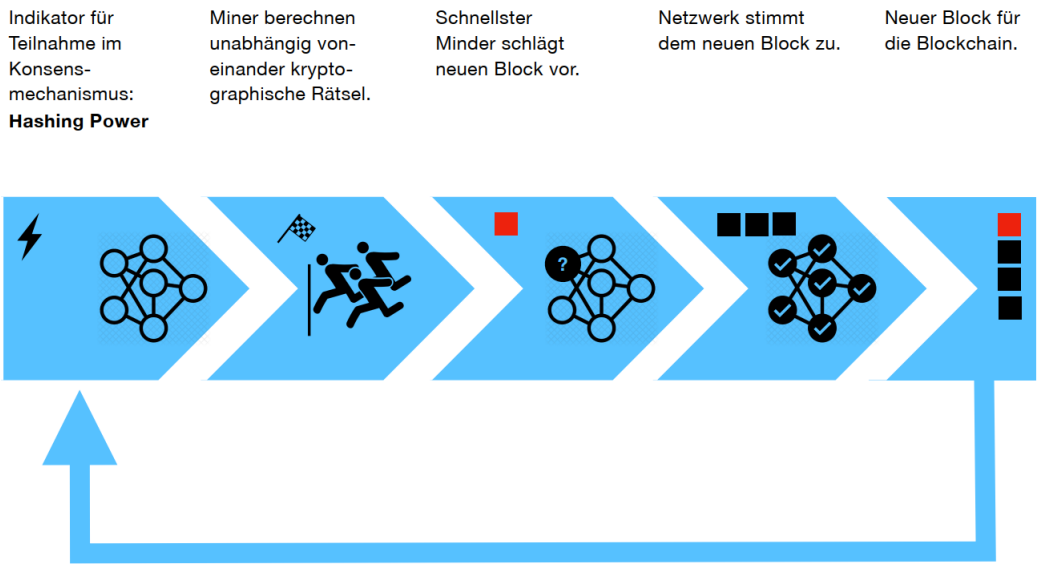


Abbildung 3.10: Funktionsschema des *Proof-of-Work*-Konsensmechanismus, eigene Darstellung nach [44].

Tangle Beim *Tangle*-Konsensmechanismus gibt es im Gegensatz zu *Proof-of-Work* keine Unterscheidung zwischen Netzwerkteilnehmern. Jeder Teilnehmer wird mit dem Auslösen einer Transaktion auch automatisch zum Miner. Bei der *Tangle*-Architektur handelt es sich um einen gerichteten azyklischen Graphen (engl.: *Directed Acyclic Graph*), der genau wie die Blockchain, eine öffentlich einsehbare Datenbank darstellt, in der alle Transaktionen des Netzwerks hinterlegt werden. Im Gegensatz zu einer länger werdenden Kette baut sich jedoch ein regellos mutendes Transaktionsnetz auf, wobei die Netzgröße durch die Anzahl verifizierter Transaktionen bestimmt wird. Das Ausführen einer Transaktion ist an

die Verifikation von mindestens zwei weiteren Transaktionen gekoppelt. Somit ist es im Interesse der Initiatoren, weitere Abläufe im Netzwerk zu prüfen.

Proof-of-Stake Der *Proof-of-Stake*-Konsensmechanismus wählt anstelle einer kryptographischen Rechenaufgabe per gewichtetem Zufall, wer einen neuen Block hinzufügen darf.[44] Vertrauen in die Glaubwürdigkeit des Teilnehmers wird durch seinen eigenen Anteil (engl.: *Stake*) an der Blockchain hergestellt. Für die Implementierung des *Proof-of-Stake*-Mechanismus werden zwei Möglichkeiten der Konsensbildung unterschieden. Bei der *Randomized block selection* basiert die Auswahl des Akteurs, der den nächsten Block validieren darf, auf dem höchsten Besitz und der Erstellung eines Hashwertes. Bei der sogenannten *coin age based selection* werden basierend auf der Dauer, die Teilnehmer bereits Anteile an der Kryptowährung halten, Rechte im Blockchain-Netzwerk gestärkt und gleichzeitig Anreize an einem langfristigen Besitz geschaffen.

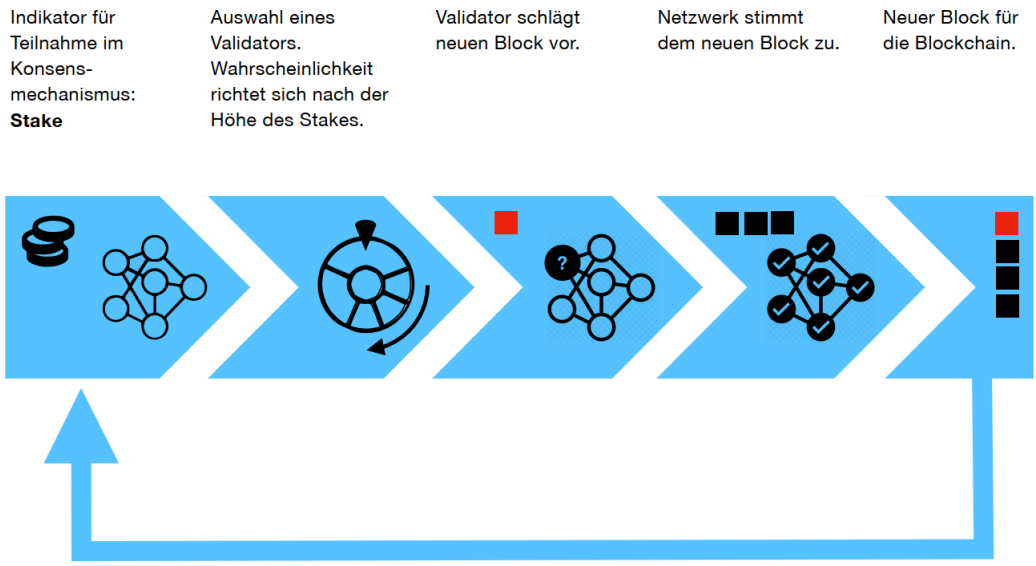


Abbildung 3.11: Funktionsschema des *Proof-of-Stake*-Konsensmechanismus, eigene Darstellung nach [44].

Proof-of-Authority Bei *Proof-of-Authority* werden Blockchains durch die validierenden Authoritäten gesichert. Die Auswahl, wer zur vertrauenswürdigen Entität wird, erfolgt willkürlich und kann nicht durch Rechenleistung oder Wertanteile beeinflusst werden. Die Teilnahme am Netzwerk ist dadurch *permissioned*, die Nutzung der Blockchain jedoch frei zugänglich. Damit stellt der *Proof-of-Authority* eine praktische und effiziente Lösung

für Blockchain-Netzwerke dar, in denen alle Teilnehmer an der Validierung bekannt und seriös sind. Der Grundgedanke ist, dass nur eine begrenzte Anzahl an Blockvalidatoren, sogenannte *authorities*, deren Identitäten dem Netzwerk bekannt sind, neue Blöcke validieren und hinzufügen dürfen. Dies macht es zu einem hoch skalierbaren System. Dadurch entfallen aufwendige Berechnungen, wodurch der Energieverbrauch erheblich reduziert werden kann.[56] Eine von der *Energy Web Foundation* für die Energiewirtschaft entwickelte Open-Source Blockchain-Plattform[8] nutzt den Algorithmus und gibt eine bis zu 30-fache Leistungsverbesserung im Transaktionsdurchsatz sowie einen deutlich geringeren Energieverbrauch im Vergleich zum *Proof-of-Authority* von *Ethereum* an.[56]

Practical Byzantine Fault Tolerance Beim *Practical Byzantine Fault Tolerance*- Konsensmechanismus wird mithilfe redundanter Abfragen eine erhöhte Toleranz gegen fehlerhafte oder manipulierte Nachrichten in einem Netzwerk aufgebaut. Ziel ist es, dass die Blockchain auch dann verlässlich arbeiten kann, wenn eine gewisse Anzahl byzantinischer Fehler auftritt. Kerngedanke ist, dass die maximale Anzahl bösartiger Knoten nicht größer oder gleich einem Drittel aller Knoten im System sein darf. Mit zunehmender Anzahl von Knoten wird das System also sicherer. Beim *Practical Byzantine Fault Tolerance*-Konsensmechanismus wechseln sich die validierungsberechtigten Akteure in der Reihenfolge, neue Blöcke zu erstellen, ab.[85]

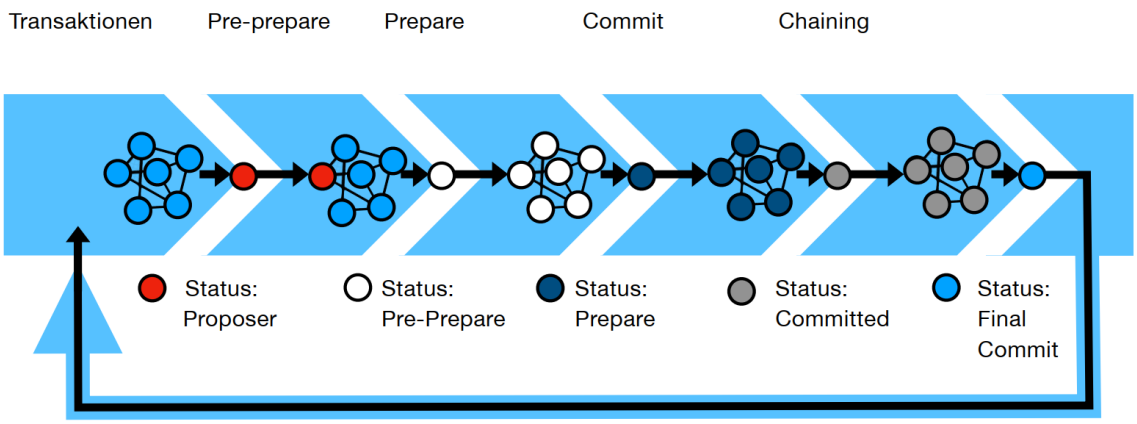


Abbildung 3.12: Funktionsschema des *Practical Byzantine Fault Tolerance*-Konsensmechanismus, eigene Darstellung nach [44].

[8] https://www.energyweb.org, Letzter Aufruf: 28.02.2021

Eine Abwandlung des *Practical Byzantine Fault Tolerance* ist der *Delegated Byzantine Fault Tolerance*-Konsensmechanismus, der zusätzlich eine Hierarchie unter allen beteiligten Akteuren vorsieht. In einem Votum werden sogenannte *professional nodes* festgelegt, denen wiederum das Validierungsrecht vorbehalten ist.[85]

Tendermint *Tendermint* baut auf dem *Proof-of-Stake*-Mechanismus auf und vereint diesen mit dem *Practical Bytantine Fault Tolerance*-Mechanismus. Bei der Validierung müssen Akteure einen *Stake* hinterlegen. Damit soll ein Anreiz geschaffen werden, weiterhin fehlerfreie Blöcke zu validieren. Die Höhe des *Stakes*, der dabei hinterlegt wird, wird als Kriterium zur Auswahl des Netzwerkknotens herangezogen.[23]

3.5 Smart Contracts

Der Begriff *Smart Contracts* wurde erstmals 1994, noch vor der Blockchain-Technologie, durch Nick Szabo im Zusammenhang mit autonomen rechtlichen Verträgen vorgestellt.[106] Seit der Entwicklung der *Ethereum*-Plattform[9] im Jahr 2015 findet das Konzept in der Praxis breite Anwendung.

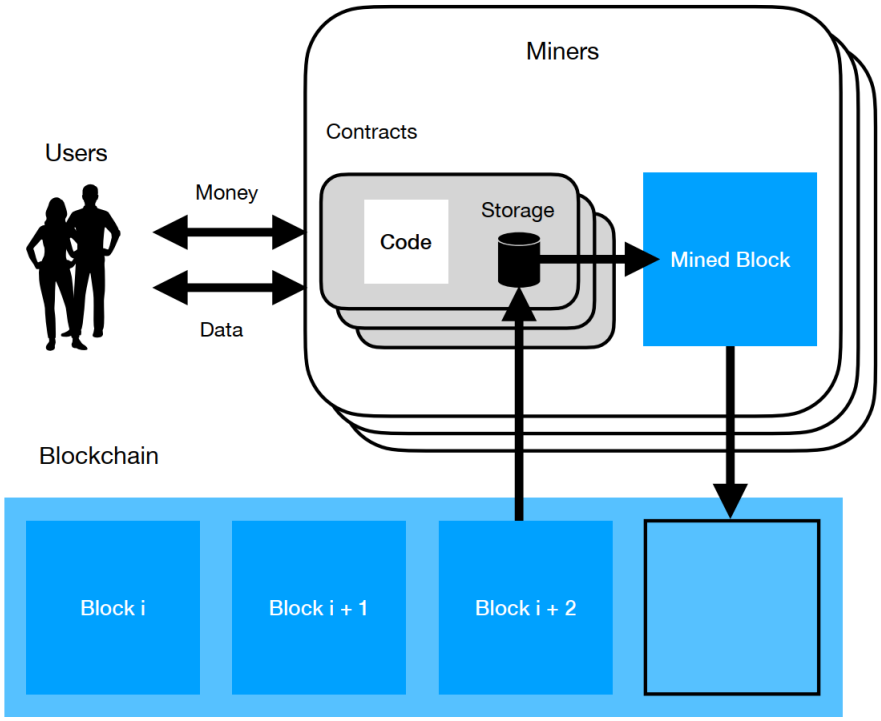


Abbildung 3.13: Schema einer dezentralen Blockchain mit *Smart Contracts*, eigene Darstellung.

[9] www.ethereum.org

Smart Contracts sind Anwendungen, die auf einem Blockchain-Protokoll basieren, im Blockchain-*Ledger* bereitgestellt und im Rahmen der Transaktionsvalidierung autonom ausgeführt werden und dadurch die Abwicklung und das Überprüfen eines virtuellen Vertrags ermöglichen. Das Ausführen der Transaktion ist vom Eintritt bestimmter Bedingungen abhängig, wodurch ein direkter Kontakt der Parteien verzichtbar wird. Hierdurch wird der Transaktionsaufwand reduziert und Kosten gesenkt. Mit Hilfe von *Smart Contracts* ist es möglich, bei Inkrafttreten programmatisch festgelegter Bedingungen und durch vollständig autonomes Durchsetzen von Vertragsklauseln zu einem Vertragsabschluss zu kommen. *Smart Contracts* können in *Opcodes*, wie der *Bitcoin*-Skriptsprache oder in höheren Programmiersprachen, wie *Solidity* oder *Java*, entwickelt werden. Nach der Entwicklung eines *Smart Contracts* auf der *Ethereum*-Plattform wird dieser in *Ethereum Virtual Machine Bytecode* kompiliert und als *Contract*-Transaktion in Blöcken gruppiert. Im Anschluss daran wird der Konsensmechanismus ausgeführt. *Ethereum* ist die am weitesten verbreitete Blockchain-Plattform für *Smart Contracts*. Die *Smart Contract*-Transaktion wird zusammen mit dem *Bytecode* an die Blockchain gesendet, damit er von allen Netzwerkteilnehmern ausgeführt werden kann. Als Ergebnis wird ein *Contract Account* für den *Smart Contract* erstellt. Der Code des *Smart Contract* kann nun über Transaktionen, welche an die Adresse des *Contract Accounts* gesendet werden, aufgerufen und ausgeführt werden. *Smart Contracts* bieten gegenüber physischen Verträgen den Vorteil, dass bei korrekter Implementierung Interpretationsschwierigkeiten der Vertragsbedingungen vermieden werden können. Gleichzeitig birgt dies jedoch die Gefahr eines *Single-Point-of-Failures*, wenn der Programmcode fehlerhaft ist. Somit hängt die Verlässlichkeit eines *Smart Contracts* derzeit noch sehr stark vom Programmierer ab und es ist schwierig, technische Hintertüren im Programmcode zu identifizieren. Aus diesem Grund werden in Kapitel acht Sicherheitsmechanismen der *Smart Contract*-Entwicklung aufgezeigt und Problemlösungsstrategien vorgeschlagen.

3.6 Fazit

In diesem Kapitel wurden die Schlüsseltechnologien und -bausteine einzeln beschrieben und deren Zusammenhang konsistent erläutert. Aus dieser Grundlagenbeschreibung lassen sich bereits Potenziale für den Einsatz in *Smart Grids* und Lösungsstrategien für einige Schwachstellen des traditionellen, zentralisierten Ansatzes erkennen. Mit dem Blockchain-basierten Ansatz kann auf zentrale Authoritäten verzichtet und so eine dezentrale *Peer-to-Peer*-Handelsplattform aufgebaut werden, die nicht mehr auf das Vertrauen in Intermediäre angewiesen ist. Ein vorab definiertes Regelwerk kann von allen Knoten des Netzwerks verteilt berechnet, überprüft und durchgesetzt werden. So entfällt zudem die Gefahr von *Single Point of Failure*-Schwachstellen im *Smart Grid*. In einem *Smart Grid* der Elektromo-

bilität werden Eintrittsbarrieren herabgesetzt und mit dem Einsatz von *Smart Contracts* wird auch der Handel von kleinsten Energiemengen ermöglicht, was wiederum Kostensenkungspotenziale im Vergleich zu konventionellen Ansätzen erwarten lässt. Die beschriebenen Grundlagen dieser Arbeit können außerdem verwendet werden, um verschiedene Blockchain-Komponenten für ein Geschäftsmodell zu bewerten und eine passende Lösung zu entwickeln.

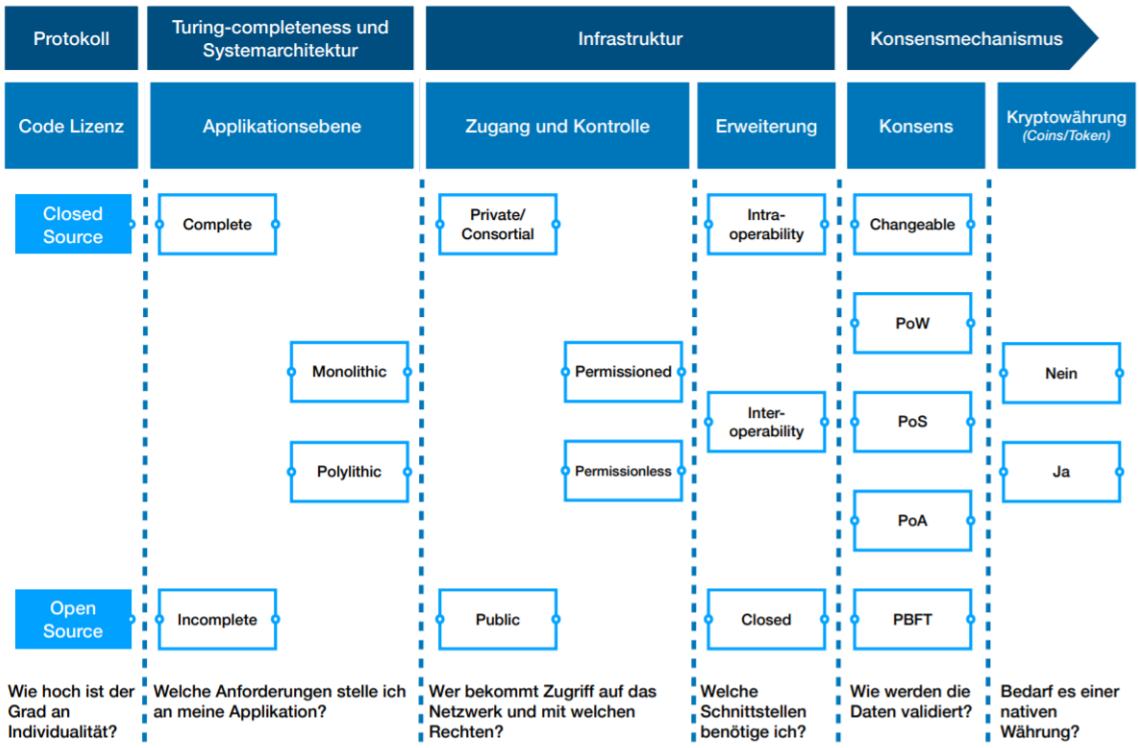


Abbildung 3.14: Überblick der Blockchain-Komponenten für die Entwicklung von Geschäftsmodellen, eigene Darstellung nach [44].

Aus den beschriebenen Grundlagen haben sich jedoch auch erste Nachteile im Vergleich zu einem zentralisierten Ansatz gezeigt. Durch die verteilte Datenhaltung der Blockchain und die Speicherung und Replikation aller Transaktionsdetails auf allen beteiligten Knoten des Netzwerks wachsen der replizierte Datenbestand und der Energie- sowie Rohstoffaufwand über den gesamten Lebenszyklus der Blockchain.[24] Im Energiesektor werden durch den Datenaustausch und die Bereitstellung von Echtzeit-Informationen enorme Datenmengen ausgetauscht. Dies zeigt bereits eines der zentralen Probleme des Blockchain-Ansatzes für das *Smart Grid*: die Skalierbarkeit.[20] Außerdem herrschen hohe Sicherheitsstandards,

hohe Ansprüche an die Performanz, Interoperabilität zwischen verschiedenartigen Systemen sowie an die IT-Sicherheit und Zuverlässigkeit. Ähnlich dem *CAP-Theorem* weist das *Blockchain-Trilemma* darauf hin, dass drei wichtige Eigenschaften eines Blockchain-Systems - die Dezentralisierung, die Sicherheit und die Skalierbarkeit - nicht perfekt nebeneinander existieren können.[20] Um die Interdependenzen der Blockchain-Merkmale für den Anwendungsfall zu vergleichen, werden im fünften Kapitel spezifische Merkmale des Blockchain-Designs herausgearbeitet und *Trade-Offs* im Kontext des Anwendungsfalls identifiziert.

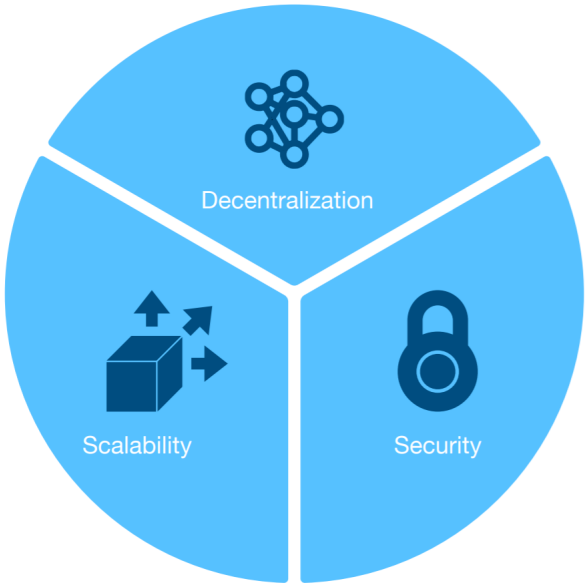


Abbildung 3.15: Sicherheit, Dezentralität und Skalierbarkeit im *Blockchain-Trilemma*, eigene Darstellung.

Im *Smart Grid* kommunizieren die Sensoreinheiten in einem offenen, drahtlosen Kommunikationskanal miteinander und die Verbraucherdaten müssen vor Netzwerkangriffen geschützt werden. Der Datenaustausch im Netzwerk öffnet einem Angreifer die Möglichkeit, Zählerstände zu ändern oder Energieeinheiten zu seinem Vorteil an verschiedene Energiequellen weiterzuleiten. Datenschutz, Vertraulichkeit und Authentifizierung sind daher von größter Bedeutung. Durch die vorgestellten Konsensmechanismen können die Stakeholder den ausgetauschten Daten im dezentralen Netzwerk ohne Intermediäre vertrauen. Dies macht *Smart Grids* unter Einsatz der Blockchain dynamischer, reduziert Wartungsprobleme und steigert die Effizienz im Vergleich zu konventionellen Ansätzen. Das Kapitel hat jedoch auch gezeigt, dass der geringe Transaktionsdurchsatz hauptsächlich auf den

verwendeten Konsensmechanismus zurückzuführen ist. Bei dem bei *Bitcoin* verwendeten *Proof-of-Work* benötigen die Knoten eine hohe Verarbeitungsleistung und -zeit, um den Algorithmus zu berechnen. Dies führt zu sogenannten Rebound-Effekten. In Kapitel fünf erfolgt daher eine Bewertung der vorgestellten Konsensmechanismen.

4 Markt- und Anforderungsanalyse

4.1 Einleitung

In diesem Kapitel werden durch eine Anwendungs- und Literaturrecherche verwandte, bereits umgesetzte Pilot- und Forschungsprojekte zusammengetragen. Daran sollen zum einen die Wirtschaftlichkeit und technologische Realisierbarkeit des Blockchain-basierten Ansatzes aufgezeigt werden und Vorteile konkreter Anwendungen in die Anforderungsanalyse einfließen. Anschließend erfolgt eine Stakeholderanalyse, die verschiedene Anspruchsgruppen identifiziert, um unterschiedliche Zielkonflikte in die Anforderungsanalyse einfließen zu lassen.

4.2 Verwandte Arbeiten

Die Wirtschaftlichkeit des Einsatzes großer stationärer Batteriesysteme zur Energiezwischenspeicherung für den Ausgleich eines Leistungsungleichgewichts wurde bereits demonstriert.[65] Das *Smart Grid*, bei dem verteilte Batterien mit volatiler Verfügbarkeit und unvorhersehbarem Ladezustand integriert werden, birgt jedoch neue Herausforderungen. Im Kontext des *Smart Grid* konnte bereits gezeigt werden, dass die Integration rückspeisefähiger Elektrofahrzeuge zur Energiespeicherung und Frequenzregelung technisch umsetzbar ist.[73] Ebenfalls wurde gezeigt, dass durch die Laststeuerung durch *Demand Side Management* die Stromnachfrage flexibilisiert und Kosten gesenkt werden können.[66] Die nachfolgenden Pilot- und Forschungsprojekte sollen die Vorteile konkreter Anwendungen aufzeigen.

Brooklyn Microgrid Das *Brooklyn Microgrid* ist eines der bekanntesten Praxisbeispiele für den Aufbau eines lokalen, dezentralen *Microgrids*.[10] Das New Yorker Pilotprojekt stellt lokal gewonnenen Strom aus erneuerbaren Energien über eine Blockchain-basierte Stromhandelsplattform den Teilnehmern zur Verfügung.[77] Das Projekt zeigt, dass die Abwicklung von Transaktionen über die Blockchain technisch realisierbar ist. Gleichermaßen zeigt das Projekt die Funktionalität von Hardware wie *Smart Metern* oder Konvertern für den Anwendungsfall. Für die nutzerfreundliche Anwendung wurde eine Applikation entwickelt, mit der Nutzer ihren Stromverbrauch tracken und Strom handeln können.

[10] Lokal abgegrenztes *Smart Grid*, das aus nur wenigen Verbrauchern und Erzeugern besteht

SonnenCommunity Ziel des Projektes ist es, dass Mitglieder einer *Community*-Plattform ihren selbst produzierten Strom untereinander handeln und mit Hilfe der Blockchain die Anlagen intelligent miteinander verknüpft werden.[67] Dezentrale Batteriespeicher werden für *Redispatch*-Maßnahmen[11] in das Stromnetz in Deutschland integriert. Das Projekt zeigt, dass ein bestehender Überschuss an Energie in einen virtuellen *Energiepool* eingespeist werden und von anderen Mitgliedern verwendet werden kann, die ihn möglicherweise benötigen, wenn die Wetterbedingungen nicht ausreichen, um genügend Energie für ihre Bedürfnisse zu erzeugen.

INEES Ziel des Projektes ist es, zu untersuchen, wie sich aus Sicht der Nutzer energiewirtschaftliche Nutzung und Mobilitätsalltag ohne Einschränkung verbinden lassen.[8] In dem Projekt wird eine Applikation als Nutzerschnittstelle entwickelt, um zu untersuchen, wie sich das persönliche Fahrverhalten der Teilnehmer des Flottenversuchs und die Anforderungen des Strommarktes miteinander verbinden lassen. Das Projekt zeigt, dass ein entwickeltes Anreizsystem die Bereitschaft der Freigabe eines Teils der Batteriekapazität für die energiewirtschaftliche Nutzung steigert.[8]

LAMP - Landau Microgrid Project Das Pilot- und Forschungsprojekt *Landau Microgrid Project* basiert auf den Erkenntnissen des *Brooklyn Microgrid*-Projekts.[111] Im Projekt werden 20 Haushalte untereinander vernetzt sowie mit *Smart Metern* ausgestattet, um über eine lokale Plattform dezentral Stromhandel betreiben zu können. Über eine App können Nutzer Preisvorstellungen angeben sowie aktuelle Messdaten abrufen. Die Transkationen für die Energieversorgung werden automatisch abgewickelt. Das Projekt untersucht insbesondere Mechanismen zur Marktpreisbildung sowie die öffentliche Akzeptanz für lokale Nachbarschafts-Energiemärkte auf Blockchain-Basis.[111]

TransActive Grid Das Pilotprojekt *TransActive Grid* integriert Erneuerbare Energieanlagen in ein *Microgrid*-Netzwerk. Es wird untersucht, wie Energie in privaten Energiespeichern gespeichert und an Energieunternehmen verkauft werden kann. So wird ein *Community*-Markt geschaffen, in dem die Teilnehmer mithilfe von *Smart Contracts* automatisiert und sicher Energie über die Blockchain kaufen und verkaufen können.[11] Das Projekt zeigt, dass in einem physischen *Smart Grid*, das vom Verteilungsnetz getrennt ist, die Kosten für Übertragungs- und Verteilungsinfrastrukturen gesenkt und der Energiepreis in Echtzeit angepasst werden kann.[11]

SMART Capital Region 2.0 Ziel des Projektes ist es, die Auswirkungen der zunehmenden Integration erneuerbarer Energien, Speicher und Lasten des Strom-, Wärme-, Gas-

[11] Eingriffe in die Erzeugungsleistung von Kraftwerken, um vor einer Überlastung zu schützen[22]

und Elektromobilitätssektors auf die Residuallast in *Smart Grids* zu untersuchen. Außerdem visualisiert das Projekt die räumliche und zeitliche Verteilung der regenerativen Erzeugung sowie der Residuallast in *Smart Grids* in einer Applikation. Damit zeigt das Projekt, wie regenerative Überschüsse regional durch den Einsatz steuerbarer Lasten und Speicher besser genutzt werden können.[103]

4.3 Stakeholderanalyse

Um die Akzeptanz und die Marktdurchdringung des Blockchain-basierten *Smart Grid* zu befördern, müssen Anforderungen mehrerer Interessensgruppen berücksichtigt werden und deren unterschiedliche Erwartungen, Bedürfnisse und Zielkonflikte in einer frühen Phase der Projektplanung in die Analyse des Anwendungsfalls einfließen.[69] Um die Anforderungen verschiedener Akteure berücksichtigen zu können, werden im Folgenden die wichtigen Stakeholder für den Anwendungsfall identifiziert und deren Ziele und Einflussfaktoren auf das Gesamtsystem abgebildet.

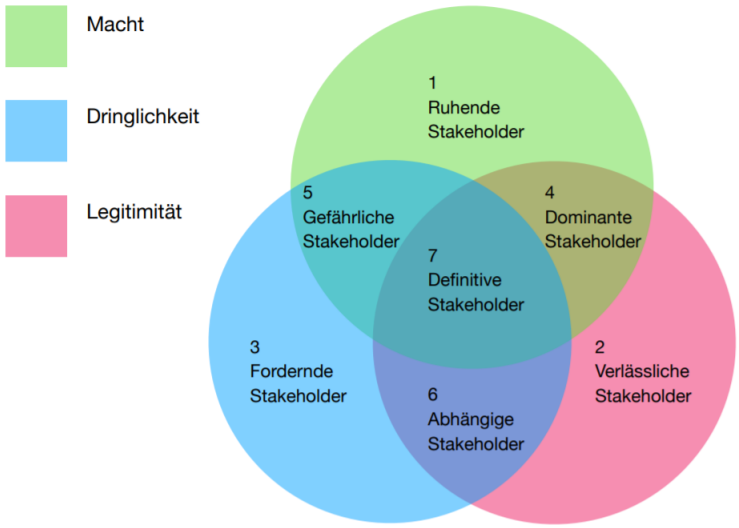


Abbildung 4.1: Stakeholder-Typologie, eigene Darstellung nach [79].

Der Stakeholder-Typologie von Mitchell, Agle und Wood[79] folgend gibt es sieben Stakeholder-Typen, die auf Basis der Kombination dreier Eigenschaften eingeteilt werden:

- Die Macht der Stakeholder, Einfluss auf das Projekt zu nehmen.
- Der Legitimität der Beziehungen und Handlungen der Stakeholder mit dem Unternehmen.

- Die Dringlichkeit des gestellten Anspruchs der Stakeholder an das Unternehmen. Ein hohes Maß an Dringlichkeit ist dann gegeben, wenn ein Stakeholder ein Anliegen als sehr wichtig und zeitkritisch erachtet.

Die Macht kann sich auf ökonomische sowie außerökonomische Ressourcen beziehen, beispielsweise Know-How oder Informationen. Die Legitimität kann juristischer, moralischer oder wissenschaftlicher Natur sein.[79] Auf je mehr Legitimitätsbasen Stakeholder sich stützen können, umso unumstrittener sind sie.

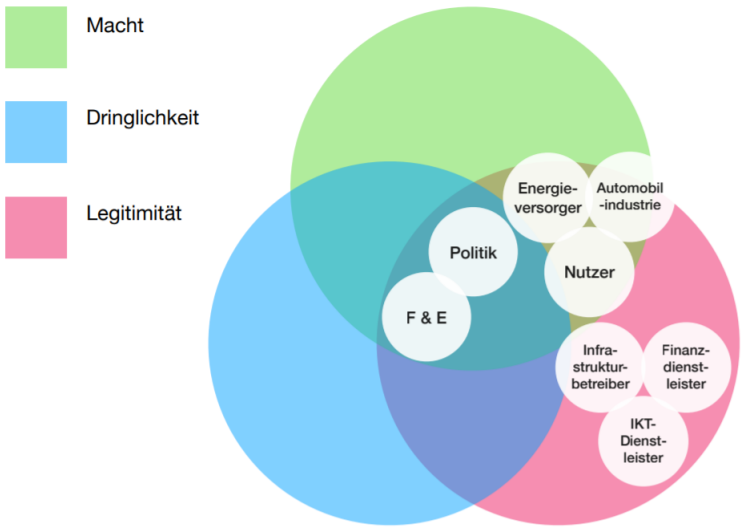


Abbildung 4.2: Konkrete Einordnung der relevanten Stakeholder, eigene Darstellung.

Den definitiven Stakeholdern werden die Politik, die Forschung und die Entwicklung zugeordnet. Politische Entscheidungsträger besitzen Zugang zu Ressourcen, zur Förderung von Forschungs- und Entwicklungsprojekten sowie die Legitimität zu Projektausschreibungen. Durch das von der Bundesregierung erklärte Ziel, bis zum Jahr 2030 mindestens sieben bis zehn Millionen Elektrofahrzeuge auf Deutschlands Straßen zu bringen, ergibt sich aus der zeitkritischen Anforderung für die Politik ebenfalls eine hohe Dringlichkeit.[83] Die Förderungsmaßnahmen der Politik sind direkt gekoppelt mit der Forschung- und Entwicklung, wodurch sich für diese die gleiche Einstufung ergibt. Als dominante Stakeholder werden die Energieversorgungsunternehmen, die Automobilindustrie sowie die Nutzer eingeordnet. Hier ergibt sich eine Überschneidung aus Macht und Legitimität. Die Nutzer erscheinen als Stakeholder, da sie die nötige Kaufkraft von Elektrofahrzeugen sowie die Nutzungsabsicht im *Smart Grid* mitbringen. Die Energieversorgungsunternehmen stellen wichtige Ressourcen, wie den Strom und zum Teil auch die Ladeinfrastruktur, zur Verfügung. Sie stehen in

direktem Geschäftskontakt zu den Nutzern. Die Automobilindustrie entwickelt und stellt die Elektrofahrzeuge, weshalb auch sie den dominanten Stakeholdern zuzuordnen sind. Als vernachlässigbare Stakeholder werden Finanzdienstleister und Infrastrukturbetreiber identifiziert. Die Legitimität ist gegeben, da sie in Geschäftskontakt mit anderen Stakeholdern stehen. Die fehlende Dimension Macht ist der Abhängigkeit der Dienstleister von den Energieversorgern und der Automobilindustrie geschuldet. Im Folgenden werden die Anforderungen der beiden Stakeholder Energieversorger und Nutzer genauer identifiziert und beschrieben.

Energieversorger Das Interesse der Energieversorger begründet sich in einem flexiblen und wirtschaftlichen Betrieb, die Anforderungen umfassen im Wesentlichen ökonomische Aspekte. Daraus lassen sich konkrete technische Anforderungen ableiten, wie die bidirektionale Anbindung mit möglichst großen Ladeleistungen und möglichst großen Batteriespeichern.[9] Eine schnelle Amortisation und eine hohe Auslastung stellen, in Abhängigkeit von Geschäftsmodellen, mögliche primäre Optimierungskriterien dar.

	Energieversorger	
SH-ES-01	Amortisationsdauer	Interesse an rentabler Ladeinfrastruktur mit schneller Amortisation
SH-ES-02	Anschaffungskosten	Kosten für die gesamte Infrastruktur für den Aufbau des *Smart Grid*
SH-ES-03	Abschreibung	Kosten unter Berücksichtung von Wertminderungen
SH-ES-04	Betriebskosten	Kosten für den laufenden Betrieb des *Smart Grid*
SH-ES-05	Verfügbarkeit	Anforderungen der Energieversorger an die Verfügbarkeit
SH-ES-06	Auslastung	Leistungsfähigkeit des *Smart Grid*
SH-ES-07	Ladezeiten	Dauer für das Be- und Entladen im *Smart Grid*
SH-ES-08	Strommenge	Menge des übertragenen Stroms in einem bestimmten Zeitintervall

Nutzer Um eine hohe Nutzerakzeptanz zu erlangen, darf die energiewirtschaftliche Nutzung der Fahrzeugbatterien die Mobilitätsanforderungen der Nutzer nicht einschränken.[29] Gleichzeitig müssen die Elektrofahrzeuge möglichst häufig mit der Ladestation verbunden werden sowie ein Teil der Batteriekapazität freigegeben werden. Dies macht das Planen zukünftiger Fahrtzeiten notwendig. Weitere Anforderung der Nutzer ist ein sicheres, kom-

fortabel zu bedienendes System. Zu den Hürden, die der Akzeptanz der Elektromobilität beim Nutzer im Wege stehen, gehören die Reichweite der Elektrofahrzeuge, die hohen Kosten für die Anschaffung und den Betrieb der Fahrzeuge sowie das Fehlen einer ausreichenden Anzahl von Ladesäulen.[88] Die Analyse verwandter Projekte, insbesondere des *INEES* und des *Brooklyn Microgrid* Projekts, haben gezeigt, dass mit der Entwicklung eines Anreizsystems Nutzer für die Freigabe von Batteriekapazitäten gewonnen werden können. Ein Vorschlag zur Ausgestaltung einer Nutzerapplikation, mit der ein Anreizsystem in ein Smart Grid integriert werden kann, wird in Kapitel 5.4 vorgestellt.

	Nutzer	
SH-US-01	Energiekosten	Interesse an rentabler Ladeinfrastruktur mit schneller Amortisation
SH-US-02	Ausfallsicherheit	Gesamte Infrastruktur umfassend
SH-US-03	Ladeinfrastruktur	einfache Bedienung und geringe Kosten
SH-US-04	Betriebskosten	Kosten für den Betrieb der häuslichen Infrastruktur
SH-US-05	User-Interface	Einfache Bedienbarkeit und sichere Schnittstellen
SH-US-06	Ladezustand anzeigen	Batteriestand muss angezeigt werden
SH-US-07	Ladevorgang starten	Auf Wunsch Starten des Ladevorgangs
SH-US-08	Abfahrtszeit planen	Fahrtzeiten und Ladevorgänge terminieren und koordinieren
SH-US-09	Mindestladezustand angeben	Angabe einer Batteriekapazität, die nicht unterschritten wird
SH-US-10	Prämienübersicht	Anzeige des aktuellen Prämienstandes

4.4 Fazit

In diesem Kapitel wurde die Wirtschaftlichkeit der Integration rückspeisefähiger Elektrofahrzeuge zur Energiespeicherung und Frequenzregelung im *Smart Grid* gezeigt. Es wurden verwandte Projekte vorgestellt, mit dem Ziel, Parallelen sowie Abgrenzungen zu dieser Arbeit aufzuzeigen. Mit dem *Brooklyn Microgrid*-Projekt wurde gezeigt, dass mit der Blockchain-Technologie Transaktionen auf einer Stromhandelsplattform zum Austausch lokal gewonnener Energie technisch umgesetzt werden können. Allerdings liegt der Fokus des Projekts nicht auf der Implementierung von *Smart Contracts*. Sowohl im *Brooklyn Microgrid*-Projekt, als auch im *INEES*-Projekt wurde eine Applikation als Nutzerschnittstelle entwickelt. Die Analyse der beiden Projekte hat gezeigt, dass die Entwicklung eines Anreiz-

systems die Bereitschaft zur Freigabe von Batteriekapazität für die energiewirtschaftliche Nutzung steigern kann.[8] Im *LAMP*-Projekt können Nutzer zusätzlich Preisvorstellungen angeben und aktuelle Batteriekapazitäten und Messdaten abrufen.[111] Die Analyse des *TransActive Grid* hat einige Parallelen zum Anwendungsfall dieser Arbeit aufgezeigt. Es setzt bei der Umsetzung auf die *Ethereum*-Blockchain und den Einsatz von *Smart Contracts*.[11] Allerdings ist der Fokus des Projektes kein *Smart Grid* der Elektromobilität. Die Implementierung von *Smart Contracts*, die *Smart Contract*-Sicherheitsanalyse und die Vorschläge für Lösungen auf bekannte Einschränkungen beim Einsatz der Blockchain heben diese Arbeit vom *Transactive Grid*-Projekt ab. Das Projekt konnte jedoch durch den Nachweis, dass Kosten für Übertragungs- und Verteilungsinfrastrukturen gesenkt werden können, den Sinn des Forschungsgegenstandes dieser Arbeit unterstreichen.[11] Es wurde gezeigt, dass es bereits Ansätze gibt, die sich mit dem lokalen Aufbau einer dezentral organisierten *Peer-to-Peer*-Handelsplattform beschäftigen. Es wurden Parallelen herausgearbeitet und Vorteile für diese Arbeit übernommen. So kann etwa die Entwicklung einer Applikation als Nutzerschnittstelle sowie die Realisierung eines Anreizsystems, welches ebenfalls über die Blockchain abgewickelt wird, eine Ausbaustufe dieser Arbeit darstellen. Dieser Ansatz wird in Kapitel 5.4 vertieft. Die Analyse hat gezeigt, dass mit dem Einsatz der Blockchain Transaktionen kostengünstig umgesetzt und selbst kleinste Strommengen ohne Intermediäre gehandelt werden können. Durch den sicheren und eindeutigen Nachweis von Transaktionen sowie die entstehende Transparenz und Flexibilität können die Dezentralisierung und Demokratisierung des Energiesystems gefördert und die einzelnen Akteure gestärkt werden. Forschungserkenntniss darüber, wie das Lademanagement, die Abrechnungs-und Authentifizierungsprozesse oder die Interoperabilität verschiedener Blockchains mit *Smart Contracts* umgesetzt werden können, gibt es kaum. Hierbei sind die technischen Herausforderungen groß, insbesondere da nur wenige regulatorische Vorgaben und Standards existieren, die sich zudem schnell verändern. Eine weitere Herausforderung ist die Ausgestaltung der Kommunikationsschnittstellen zu den verschiedenen Stakeholdern. Daher wurden die Stakeholder in diesem Kapitel identifiziert und deren Interessen herausgearbeitet. Die Analyse hat gezeigt, dass es bereits verwandte Forschungsprojekte gibt, jedoch noch viele offene Fragen bezüglich der infrastrukturellen Ausgestaltung der Blockchain bestehen. Im folgenden Kapitel werden Merkmale der Netzwerkarchitektur identifiziert und bewertet, um daraus wiederum eine bestmögliche Blockchain-Architektur identifizieren und eine *Smart Contract*-Programmiersprache auswählen zu können. Im Energiesektor herrschen hohe Standards an Datenschutz und Datensicherheit. Da mit dem in dieser Arbeit verfolgten Ansatz perspektivisch alle Akteure miteinander im *Smart Grid* verbunden sind, stellt der Schutz der Infrastruktur eine große Herausforderung dar, auf die die betrachteten Projekte nicht öffentlich eingehen. Es bedarf daher einer eingehenden

Untersuchung der Auswirkungen einer Blockchain-basierten und dezentralen *Peer-to-Peer*-Energiehandelsplattform auf den Schutz kritischer Infrastrukturen. In Kapitel 8 wird diese Überprüfung mit einer *Smart Contract*-Sicherheitsanalyse angestoßen.

5 Entwurf

5.1 Einleitung

In Kapitel drei wurden die Schlüsseltechnologien und -bausteine der Blockchain einzeln beschrieben und deren Zusammenhang erläutert. Aus dieser Grundlagenbeschreibung lassen sich Potenziale erkennen und Lösungsstrategien für einige Schwachstellen eines traditionellen, zentralisierten *Smart Grid*-Ansatzes ablesen. Um die Auswahl einer geeigneten Blockchain-Netzwerkarchitektur für einen Geschäfts- und Anwendungsfall zu erleichtern, werden in diesem Kapitel auf Basis der erlangten Erkenntnisse kontextabhängige Merkmale der Netzwerkarchitektur und des Designs der Blockchain herausgearbeitet und individuell für den Anwendungsfall bewertet. Diese Bewertung bildet die Grundlage für die Begründung der Plattformentscheidung. Außerdem hat die Marktanalyse in Kapitel vier gezeigt, dass die Konzeptionierung eines Anreizsystems, das einen hohen Beitrag zur energiewirtschaftlichen Optimierung vergütet, die Bereitschaft der Freigabe eines Teils der Batteriekapazität für die energiewirtschaftliche Nutzung steigern kann.[77] Aus diesem Grund wird ein prototypisches Anreizsystem vorgestellt.

5.2 Netzwerkarchitektur

Die Auswahl einer Blockchain-Plattform ist maßgebend für das gesamte Geschäftsmodell und bildet die Grundlage für die Auswahl geeigneter *Smart Contract*-Programmiersprachen, Konsensmechanismen, der Programmierumgebung und kryptographischer Verfahren. Dies muss entsprechend bereits bei der Konzeption berücksichtigt werden, da es schwierig ist, die Blockchain post hoc anzupassen. Im Folgenden werden Merkmale des Blockchain-Designs identifiziert und Interdependenzen zwischen ihnen im Kontext des *Smart Grid* aufgezeigt. Aus diesen Merkmalen begründet sich anschließend die Plattformentscheidung.

	Sicherheit	
DLT-SEC-01	Verfügbarkeit	Wahrscheinlichkeit, dass bei Bedarf auf das *Smart Grid* zugegriffen werden kann
DLT-SEC-02	Konsistenz	Alle Knoten speichern dieselben widerspruchsfreien Daten vollständig im *Ledger*
DLT-SEC-03	Datenintegrität	Informationen sind vor unbefugten Änderungen und löschen ausreichend geschützt
DLT-SEC-04	Vertraulichkeit	Unbefugter Zugriff auf Vertrags- oder Verbrauchsdaten wird verhindert
DLT-SEC-05	Verschlüsselungsstufe	Sicherheitsstufe bspw. bei Authentifizierungen oder dem Erstellen von Schlüsselpaaren
DLT-SEC-06	Vertrauen in Knotenpunkte	Vertrauenswürdigkeit der einzelnen *Nodes* im *Smart Grid*
DLT-SEC-07	Forks	Wahrscheinlichkeit, dass unterschiedliche Verzweigungen der Mainchain existieren
DLT-SEC-08	Fehlertoleranz	Eigenschaft, Funktionsweise aufrecht zu erhalten, wenn Fehler auftreten
DLT-SEC-09	Resilienz	Die Fähigkeit, nach dem Eintreten eines Ereignisses zu einem Zustand zurückzukehren
DLT-SEC-10	Verwundbarkeit	Verwundbarkeit gegenüber gezielten Angriffen auf das *Smart Grid*
DLT-SEC-11	Nachweisbarkeit	Sicherheit, dass eine Signatur oder Nachricht wirklich von einem bestimmten Akteur stammt
DLT-SEC-12	Partitionstoleranz	Das System arbeitet weiter, wenn *Nodes* im *Smart Grid* nicht mehr miteinander kommunizieren

	Performanz	
DLT-PER-01	Transaktionsdurchsatz	Anzahl der Transaktionen, die in einem bestimmten Zeitintervall validiert werden
DLT-PER-02	Energieeffizienz	Abhängig insbesondere von der Effizienz des Konsensmechanismus
DLT-PER-03	Grad der Dezentralisierung	Anzahl der Knoten, die an Transaktionsvalidierung teilnehmen
DLT-PER-04	Blockgröße	Datenmenge, die in einem Block gespeichert werden kann
DLT-PER-05	Persistierungslatenz	Zeit zwischen Einreichen einer Transaktion und Zeitpunkt, an dem jeder *Node* im *Smart Grid* die Transaktion erhalten hat
DLT-PER-06	Skalierbarkeit	Fähigkeit der Blockchain, zunehmende Arbeitsbelastung im *Smart Grid* zu bewältigen
DLT-PER-07	Geschwindigkeit des Konsensmechanismus	Für die Validierung neuer Blöcke erforderliche Dauer
DLT-PER-08	Systemkomplexität	Die Komplexität des gesamten *Smart Grid*
DLT-PER-09	Erforderliche Bandbreite	Die Bandbreite, die für den Datenaustausch benötigt wird

Für ein Blockchain-basiertes *Smart Grid* sollte eine hohe Verfügbarkeit, Ausfallsicherheit und Widerstandsfähigkeit gegen typische Gefahren durch eine ausreichende Anzahl von Knoten im Netzwerk erreicht werden. Dies bedeutet gleichzeitig einen geringeren Transaktionsdurchsatz. Dieser kann jedoch durch die Auswahl eines geeigneten Konsensmechanismus verbessert werden. Zudem sollte die Blockgröße für Anwendungen im Energiesektor so gewählt werden, dass Echtzeitdaten übermittelt werden können. Die Vergrößerung der Blockgröße führt zur Verzögerung der Blockausbreitung im Netzwerk, was wiederum zu einem längeren inkonsistenten Zustand zwischen den Knoten führt. *Bitcoin* erstellt alle 10 Minuten einen neuen Block und hat eine feste maximale Blockgröße von 1 MB.[78] *Ethereum* erstellt alle 17 Sekunden neue Blöcke, die Blockgröße wird von den Knoten individuell festgelegt. Die Verzögerungen bei der Blockausbreitung führen gleichzeitig zu einer Erhöhung der erforderlichen Bandbreite und der Wahrscheinlichkeit erfolgreicher Mining-Angriffe.[93] Je mehr Knoten am Netzwerk teilnehmen, desto größer ist die Anonymität der Benutzer. Eine öffentliche, *unpermissioned* Blockchain, wie *Ethereum*, verspricht daher mehr Anonymität. Im Gegensatz dazu bietet ein kleineres Netzwerk mit verifizierten und

identifizierbaren Knoten einen höheren Transaktionsdurchsatz, da schnellere Konsensalgorithmen verwendet werden können.[12] Im Gegensatz dazu hat eine Blockchain mit einem hohen Durchsatz, wie *HyperLedger-Fabric* mit bis zu 3.500 Transaktionen pro Sekunde, eine schlechte Skalierbarkeit und unterstützt nur eine kleine Anzahl von Knoten, was die Verfügbarkeit und Ausfallsicherheit im Vergleich zu *Bitcoin* oder *Ethereum* reduziert und für ein *Smart Grid* ungeeignet ist. Umso mehr Programmiersprachen unterstützt werden, desto mehr Möglichkeiten haben Angreifer, Exploits auszunutzen.[13] Je mehr Funktionen im *Smart Contract* implementiert werden, desto höher wird seine Komplexität. Dies hat Auswirkungen auf die Ausführungszeit. Mit der Komplexität des *Smart Contract* steigen also auch Faktoren wie der Energieverbrauch und die Transaktionsvalidierung. Die Analyse der Netzwerkarchitektur hat gezeigt, dass bestimmte Eigenschaften entweder komplementär oder widersprüchlich sind.[30] Dies geht auch aus dem *CAP-Theorem* hervor, das besagt, dass in verteilten Systemen lediglich zwei der drei Eigenschaften Konsistenz, Verfügbarkeit und Ausfalltoleranz gewährleistet sein können.[15] Die aus den Merkmalen resultierenden Interdependenzen im Kontext des Anwendungsfalls sind Grundlage für die Plattformentscheidung.

	Usability	
DLT-US-01	Interoperabilität	Fähigkeit zum Austausch zwischen verschiedenen Blockchains und Kommunikation mit externen Diensten aus dem *Smart Grid*
DLT-US-02	Modularität	Möglichkeit, einzelne Module der Blockchain, beispielsweise den Konsensmechanismus, auszutauschen
DLT-US-03	Zugriff auf *Smart Grid*	Die Möglichkeit, am *Smart Grid* teilzunehmen
DLT-US-04	Leichte Bedienbarkeit	Die Möglichkeit, einfach auf das *Smart Grid* zuzugreifen und damit zu arbeiten
DLT-US-05	Unterstützung für unterschiedliche Akteure	Das Ausmaß, in dem verschiedene Akteure, bspw. Prosumer, am *Smart Grid* teilnehmen können
DLT-US-06	Nutzerschnittstelle	Über eine Applikation kann auf das *Smart Grid* zugegriffen werden

[12] Wie der *Practical Byzantine Fault Tolerance*-Konsensmechanismus

[13] Systematisches Ausnutzen von Schwachstellen, die bei der Entwicklung eines Programms entstanden sind

5.3 Konsensmechanismus

Auf Grundlage der Merkmale im Blockchain-Design konnten für den Energiesektor insbesondere hohe Sicherheitsstandards, hohe Ansprüche an die Performanz, die Interoperabilität zwischen verschiedenartigen Systemen sowie die IT-Sicherheit und die Zuverlässigkeit des Gesamtsystems als wichtigste Merkmale identifiziert werden. Gleichzeitig werden große Datenmengen ausgetauscht.

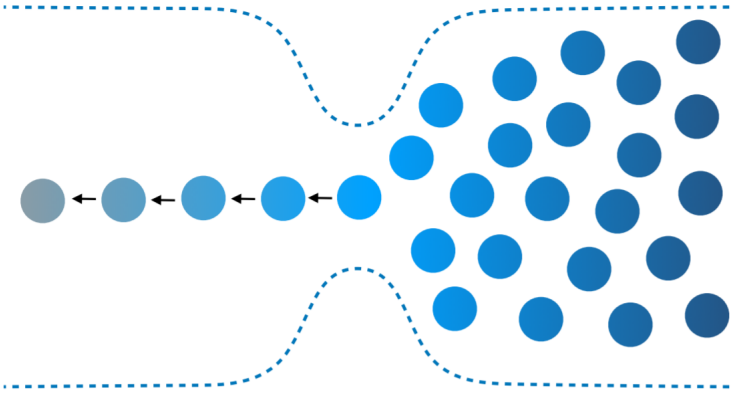


Abbildung 5.1: Engpass in der Validierung beim *Proof-of-Work*, eigene Darstellung.

Dies zeigt bereits eines der zentralen Probleme des Blockchain Ansatzes: die Skalierbarkeit. Dabei hängt der Transaktionsdurchsatz hauptsächlich vom verwendeten Konsensmechanismus ab. Dieser spielt eine entscheidende Rolle bei der Aufrechterhaltung der Sicherheit und Effizienz der Blockchain. Die Verwendung eines geeigneten Algorithmus kann die Leistung der Blockchain-Anwendung erheblich steigern.

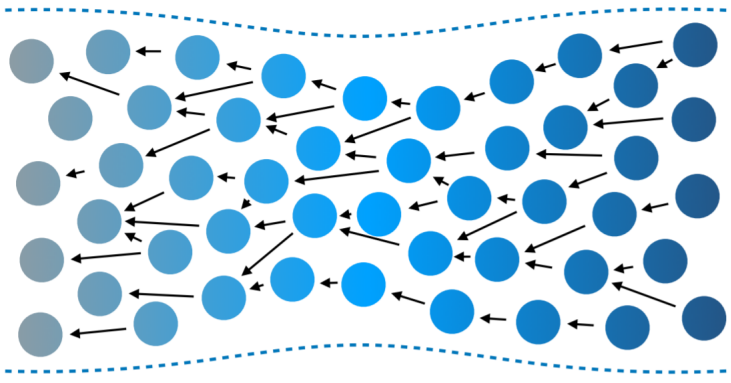


Abbildung 5.2: Validierung beim *Tangle*-Prinzip, eigene Darstellung.

Die Analyse der verschiedenen Algorithmen hat gezeigt, dass die Knoten beim *Proof-of-*

Work eine hohe Verarbeitungsleistung und -zeit benötigen, um den Algorithmus zu berechnen.[32] Einige vorgestellte Konsensmechanismen zeigen signifikante Vorteile im Transaktionsdurchsatz. So ist beim *Proof-of-Stake* die rechnerische Komplexität des kryptografischen Rätsels deutlich geringer und in der Regel unempfindlich gegenüber der Größe des Netzwerks, daher ist er auch für große Systeme sehr energieeffizient, wodurch der Strombedarf deutlich gesenkt werden kann. *Ethereum* versucht daher, von *Proof-of-Work* auf *Proof-of-Stake* umzustellen.[55] Der *Proof-of-Stake*-Mechanismus erlaubt eine ressourcensparendere Berechnung des Konsens und verringert dadurch den Stromeinsatz zur Validierung von Blöcken deutlich.[32] Zusätzlich hat die Auswahl des Konsensmechanismus Auswirkung auf die Sicherheit des Gesamtsystems. Beispielsweise sinkt die Gefahr von 51%-Attacken beim Proof-of-Stake, da sich Angreifer durch die Verluste eigener *Stakes* selbst schädigen. Für den Aufbau eines *Smart Grid* wird daher der *Proof-of-Stake*-Mechanismus als klar effizienterer Konsensmechanismus für die Implementierung von *Smart Contracts* betrachtet.

5.4 Anreizsystem

Die Analyse verwandter Projekte und die daraus identifizierten Anforderungen haben gezeigt, dass der Beitrag eines einzelnen Elektrofahrzeugs zur energiewirtschaftlichen Einbindung stark abhängig ist von der Fahrzeugnutzung und den freigegebenen Ladekapazitäten.

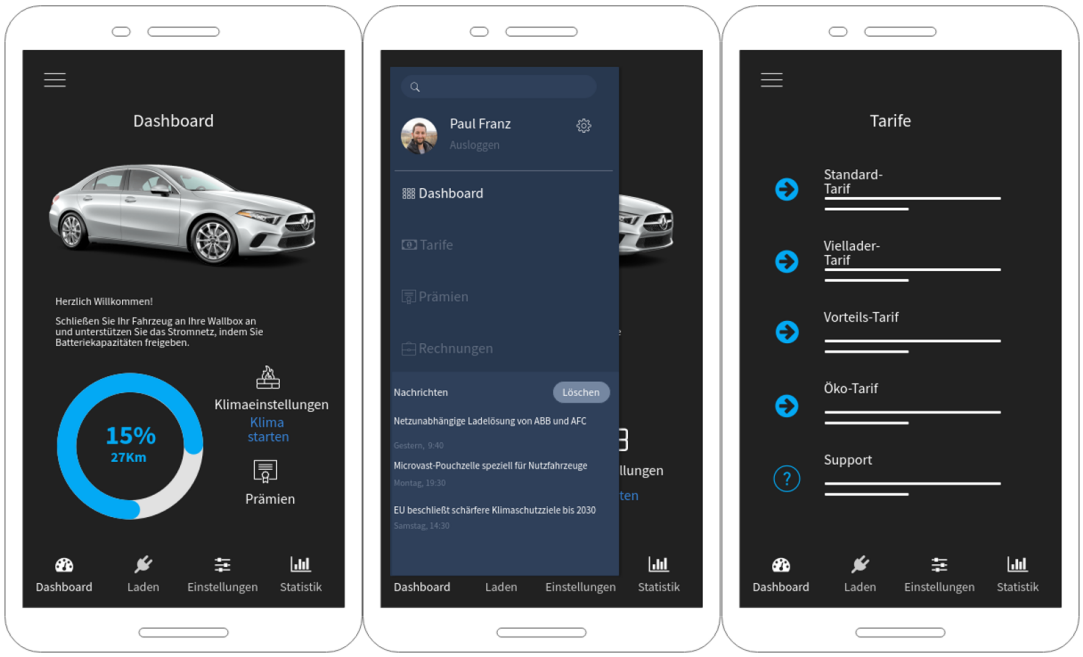


Abbildung 5.3: Hauptmenü, Dashboard und Tarifauswahl einer Nutzerapplikation, eigene Darstellung.

So bietet ein Fahrzeug, das ausschließlich zum Nachladen an die Ladestation angeschlossen wird, kein Potenzial zur energiewirtschaftlichen Optimierung. Durch die Freigabe von Fahrzeugbatteriekapazitäten zur Unterstützung des Stromnetzes dürfen grundsätzlich keine Einschränkungen in der alltäglichen Mobilität der Nutzer entstehen.

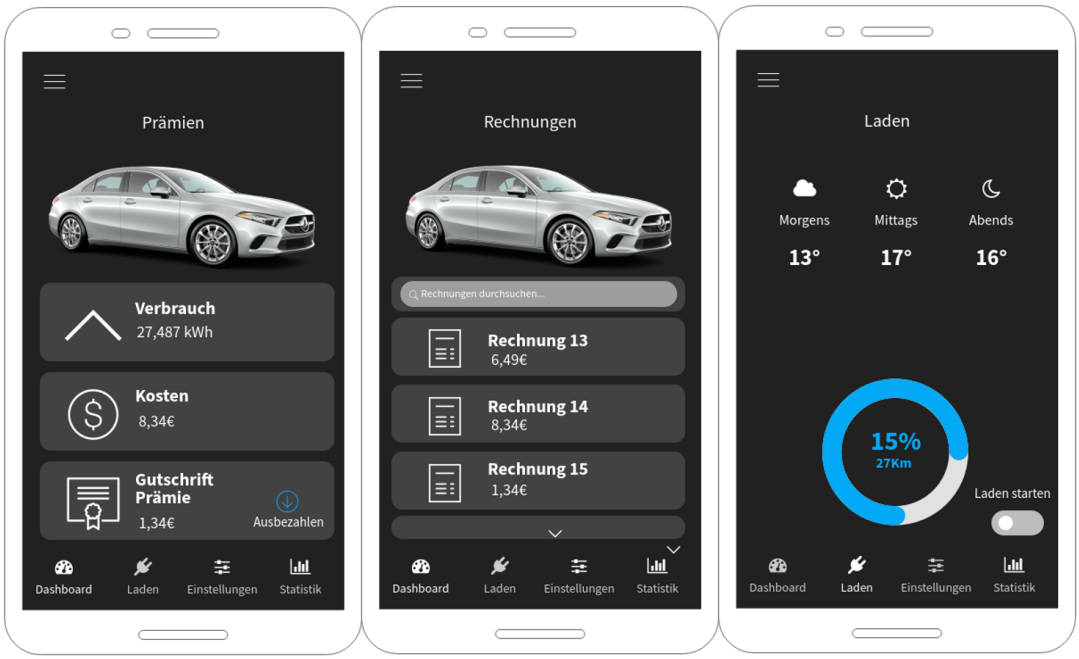


Abbildung 5.4: Prämien, Rechnungen und Ladefortschritt einer Nutzerapplikation, eigene Darstellung.

Optimal ist es, das Fahrzeug während jeder Standzeit mit der Ladestation zu verbinden, einen geringen Mindestladezustand einzustellen und die Ladung zur nächsten Abfahrt über eine in einer Schnittstelle hinterlegte Abfahrtszeit zu programmieren. Grundlage des Anreizsystems sind darüber hinaus die aus der Stakeholder- und Anforderungsanalyse gewonnenen funktionalen sowie nichtfunktionalen Anforderungen der Nutzer und Energieversorger. Nachfolgend werden diese Anforderungen akkumuliert und in einem Vorschlag zur Ausgestaltung einer Nutzerapplikation zusammengefasst. Über die vorgeschlagene Applikation können Nutzer ihren aktuellen Batteriestand anzeigen lassen. Abhängig vom Batteriestand kann die gewünschte Ladekapazität festgelegt und überschüssige Energie freigegeben werden. Dafür erhalten Nutzer Prämien, die sich ebenfalls über die Applikation einsehen und abbuchen lassen. Für die Integration der Applikation in die *Ethereum*-Blockchain muss das Frontend über eine *API* mit der Blockchain verbunden werden. Für die Verbindung mit einer Android Applikation wird die *web3.js-API* vorgeschlagen.

Abbildung 5.5: Ladestand-, Abfahrtzeitenplaner und Statistik einer Nutzerapplikation, eigene Darstellung.

Web3j ist eine modulare Sammlung von Java- und Android-Bibliotheken, mit der über *HTTP*, *IPC* oder *Websockets* mit *Smart Contracts* und dem *Ethereum*-Netzwerk kommuniziert werden kann. Über *MetaMask* oder einem *Web3*-Anbieter wie *Ganache* kann mit jedem *Ethereum*-Netzwerk interagiert werden. *MetaMask* fügt die *Ethereum web3-API* in den *Javascript*-Kontext der Anwendung ein, sodass sie aus der Blockchain lesen kann.

5.5 Plattformentscheidung

Nachfolgend werden drei Blockchain-Protokolle - *Hyperledger Fabric*, *Corda* und *Ethereum* - in Kürze vorgestellt und im Hinblick auf ihr Potenzial für den Anwendungsfall miteinander verglichen. Alle drei ermöglichen die Ausführung von *Smart Contracts*.

Hyperleger Fabric *Hyperledger Fabric* wurde für die Entwicklung von Businessapplikationen auf einer privaten und zugangsbeschränkten Blockchain entwickelt.[40] Während die grundlegende Struktur und bestimmte Standards vorgegeben sind, um die Interoperabilität und Konsistenz zu gewährleisten, ist die Entwicklung von Anwendungen modular aufgebaut. Einzelne Komponenten, wie etwa der Konsensmechanismus, können frei gewählt werden. So wird ein hoher Grad an Individualisierbarkeit ermöglicht.[40] Einen potenziellen Vorteil in einem *Smart Grid* gegenüber anderen Protokollen bietet der Einsatz von *Channeln*. Ein *Hyperledger Fabric Channel* ist ein privates *Subnetz* für die Kommunikation

zwischen zwei oder mehreren Netzwerkmitgliedern, um private und vertrauliche Transaktionen durchzuführen. Da jede Partei authentifiziert und autorisiert sein muss, um auf dem *Channel* Transaktionen durchzuführen, können die Transaktionsinformationen nur von den jeweiligen *Channel*-Teilnehmern, nicht aber dem gesamten Netzwerk, eingesehen werden. Zusätzlich nutzt *Hyperledger Fabric* das *Hardware Security Module*, um digitale Schlüssel zu verwalten.[50] Damit könnte das Identitätsmanagement im *Smart Grid* verbessert und der Schutz sensibler Daten erhöht werden. Da nur ausgewählte Teilnehmer eines Netzwerks oder *Channels*, sogenannte *Endorsing Peers*, die Verifikation von Transaktionen übernehmen und nicht alle Netzwerk- oder Channel-Teilnehmer an diesem Prozess beteiligt sind, bietet *Hyperledger Fabric* einen hohen Grad an Flexibilität und eine insgesamt hohe Skalierbarkeit, jedoch ist es auch anfälliger gegenüber Angriffen und hat eine potenziell geringere Datenintegrität und Systemsicherheit.

Ethereum *Ethereum* ist ein Blockchain-Protokoll für die Implementierung von *Smart Contracts*. Mit der Kryptowährung *Ether* verfügt *Ethereum* über die zweitgrößte Kryptowährung. *Ethereum* bietet die Möglichkeit, generischen Code und somit Programme jeder Art in der Blockchain zu implementieren. Als Laufzeitumgebung dient die *Ethereum Virtual Machine*. *Ethereum* hat eine breite Entwicklergemeinschaft, ausgereifte Sicherheitsanalysetools und eine gute Dokumentation. Daher wurde *Ethereum* für die Implementierung von *Smart Contracts* ausgewählt.

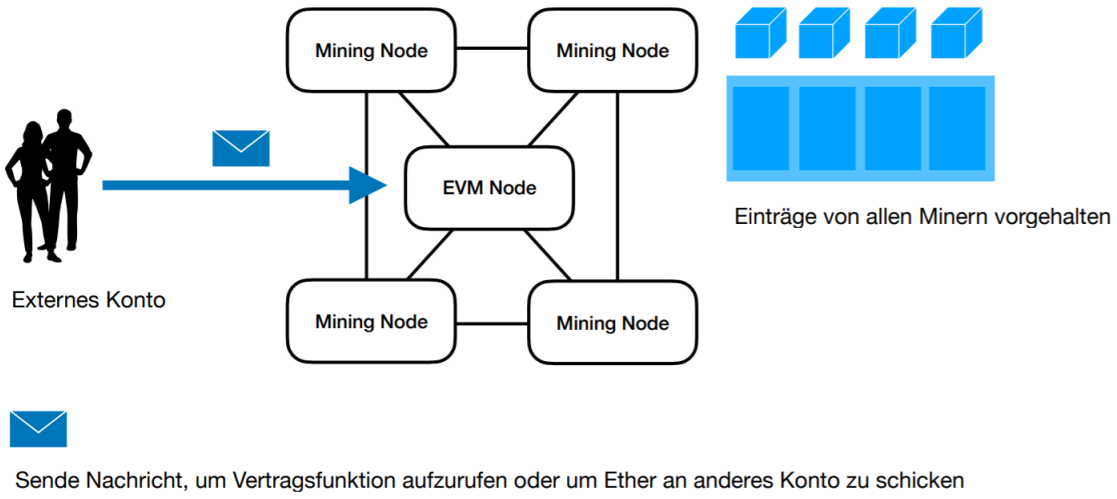


Abbildung 5.6: Die Komponenten der *Ethereum*-Plattform, eigene Darstellung.

Corda *Corda* ist ein privates und zugangsbeschränktes Blockchain-Protokoll, das eine Kryptowährung über *Smart Contracts* implementieren kann. Im Gegensatz zu *Hyperledger Fabric*[14] erhalten bei Corda lediglich die an der Transaktion direkt beteiligten Akteure Zugriff auf Daten.[36] Vertrauliche Transaktionsdetails werden so zusätzlich geschützt. Zur Verifizierung von Transaktionen setzt *Corda* auf *Notary Clusters*, nicht auf *Endorsing Peers*. *Notary Clusters* stellen sicher, dass die Transaktionen einzigartig und valide sind. Das Autorisieren wird von den an der Transaktion beteiligten Akteuren selbst vorgenommen, wofür diese Einblick in die Transaktionshistorie erhalten. Neben einer guten Skalierbarkeit und einer guten Durchsatzgeschwindigkeit durch die Verifikation auf Transaktionsebene liegt in der Einsicht in die Transaktionshistorie allerdings eine potenzielle Schwachstelle im Hinblick auf den Datenschutz im *Smart Grid*.

5.6 Fazit

In den vorigen Kapiteln konnten bereits einige Hürden bei der Umsetzung eines Blockchain-basierten *Smart Grid* identifiziert werden. Dazu gehören insbesondere Probleme bei der Skalierbarkeit, der mangelnden Interoperabilität zwischen Blockchain-Netzwerken, fehlende Standards bei der Kommunikation der Einzelsysteme, der mangelnde Transaktionsdurchsatz und der hohe Energieverbrauch einzelner Konsensmechanismen. In diesem Kapitel wurden verschiedene Plattformen vorgestellt und im Kontext des Anwendungsfalls bewertet, um die Auswahl einer geeigneten Blockchain-Netzwerkarchitektur zu erleichtern. Die Analyse ergab, dass die *Ethereum*-Blockchain für die prototypische Implementierung und für die Integration von *Smart Contracts* in das *Smart Grid* geeignet ist.

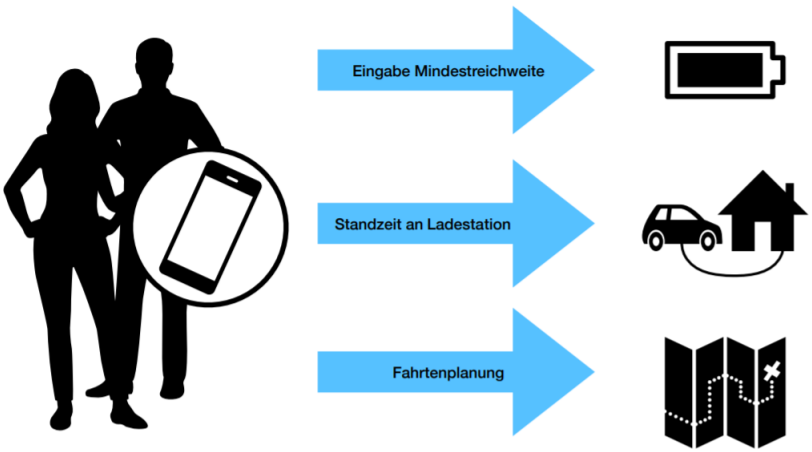


Abbildung 5.7: Parameter des vorgestellten Anreizsystems, eigene Darstellung.

[14] Informationszugriff ist auf die Teilnehmer eines *Channels* beschränkt

Aus der Marktanalyse in Kapitel 4 ging hervor, dass ein Anreizsystems einen wichtigen Beitrag zur Nutzerakzeptanz leisten kann.[77] Daher wurde in diesem Kapitel ein prototypisches Anreizsystem vorgestellt. Mit der Konzeptionierung des Anreizsystems wurde gezeigt, wie ein hoher Beitrag zur energiewirtschaftlichen Optimierung vergütet wird. Somit kann die Bereitschaft der Freigabe eines Teils der Batteriekapazität für die energiewirtschaftliche Nutzung gesteigert werden.[77] Das folgende Kapitel widmet sich einer detaillierten Beschreibung der *Ethereum*-Plattform und der Programmiersprache *Solidity*.

6 Ethereum

6.1 Einleitung

In diesem Kapitel erfolgt eine Einführung in die Grundlagen der *Ethereum*-Blockchain und der *Ethereum Virtual Machine*. Es werden Vor- und Nachteile der Laufzeitumgebung für den Anwendungsfall betrachtet sowie abschließend die Programmiersprache *Solidity* syntaktisch beschrieben. Dabei werden die wichtigsten Bausteine vorgestellt, die im weiteren Verlauf für ein gutes Verständnis der Implementierung von Bedeutung sind. Die Einführung stützt sich dabei teilweise auf die offizielle Dokumentation von *Ethereum*.[61]

6.2 Ethereum Virtual Machine

Die *Ethereum Virtual Machine* ist die Laufzeitumgebung im *Ethereum*-Netzwerk, die Programme, *Smart Contracts* und *DApps (Decentralized Applications)* ausführt.

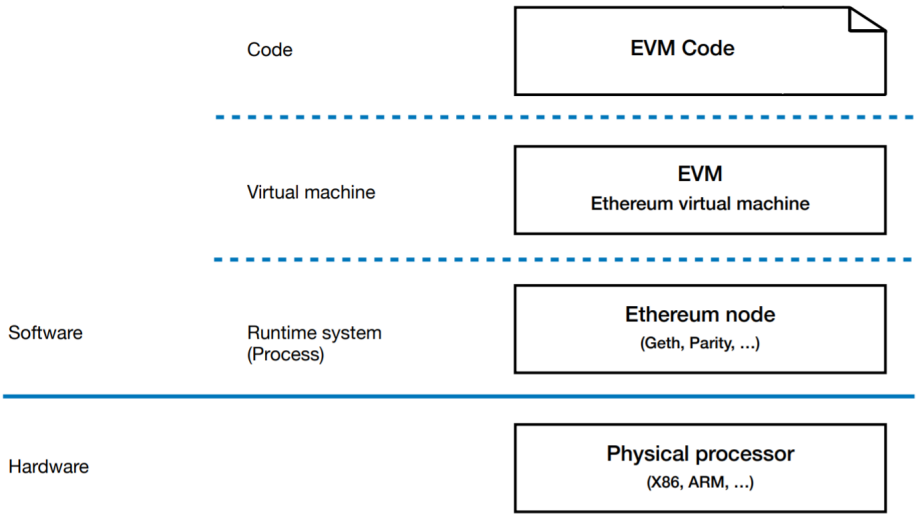


Abbildung 6.1: *EVM* im *Ethereum*-Netzwerk, eigene Darstellung nach [98].

Sie baut auf einer *Stack*-basierten Sprache mit einem vordefinierten *Opcode*-Befehlsatz auf. Ein *Smart Contract* besteht im Wesentlichen aus einer Reihe von *Opcode*-Anweisungen, die von der *Ethereum Virtual Machine* nacheinander ausgeführt werden.[26] Die Grafik zeigt die Verortung der *Ethereum Virtual Machine* im *Ethereum*-Ökosystem. Die *Ethereum Virtual Machine* erzeugt somit eine Abstraktionsebene zwischen dem ausführbaren Code und der ausführenden Maschine.

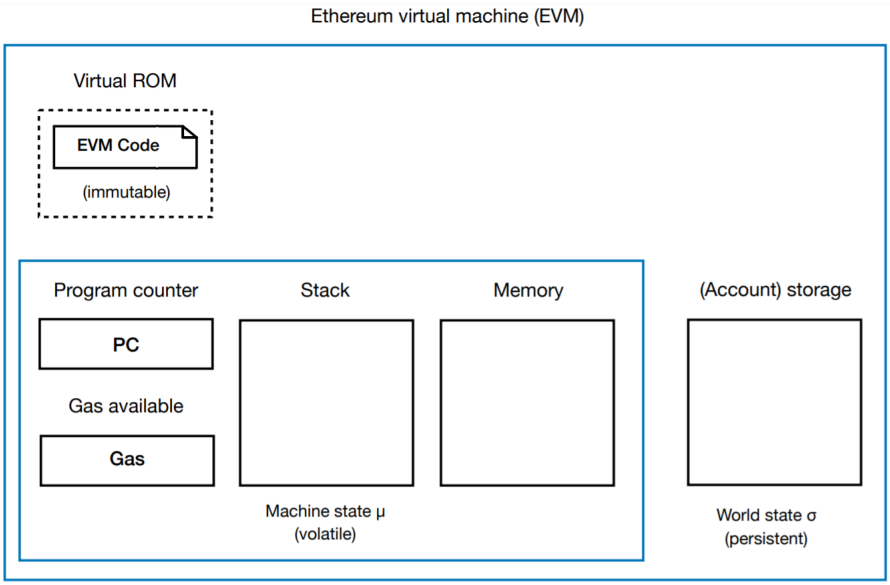


Abbildung 6.2: Architektur der *EVM*, eigene Darstellung nach [54].

Damit wird die Portabilität von Software verbessert und gleichzeitig sichergestellt, dass Anwendungen voneinander und von ihrem *Host* isoliert sind, sodass *Smart Contracts* nicht direkt auf Prozesse, das Netzwerk oder das Dateisystem zugreifen.

Accounts Wichtiger Bestandteil der *Ethereum*-Blockchain sind *accounts*. *Accounts* besitzen einen Status und eine Adresse, durch die sie identifiziert und referenziert werden können. *Accounts* können *Ether* senden und empfangen und mit *Smart Contracts* interagieren. Der *account state* beinhaltet den *nonce* zur Beschreibung der Anzahl der gesendeten Transaktionen, die *balance* zur Angabe der Menge an hinterlegtem *Ether*, den *storageRoot* für die Lokalisierung des Speichers und den *codeHash*, eine Referenz in die seperate Datenbank, in der der zur Adresse gehörende *Bytecode* persistiert wird. Man unterscheidet zwischen zwei verschiedenen *account*-Typen.

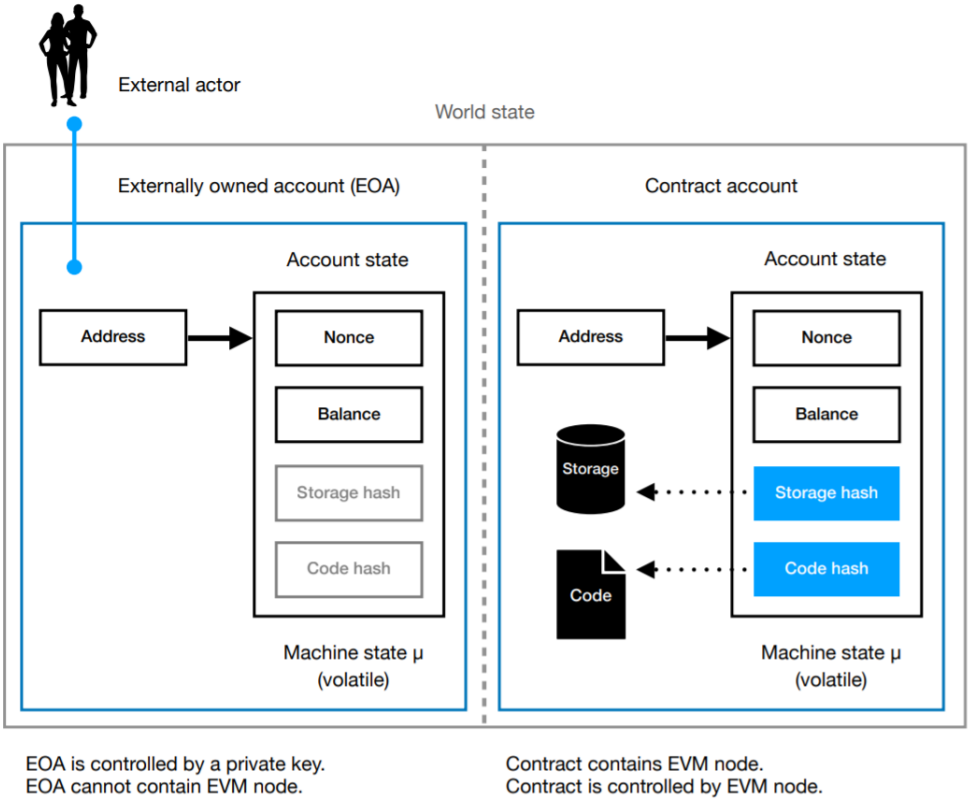


Abbildung 6.3: Die zwei *account*-Typen in *Ethereum*, eigene Darstellung nach [98].

Externally Owned Accounts *Externally Owned Accounts* sind eine Kombination von öffentlicher Adresse und *Public Key*. Sie können eigenständig Transaktionen initiieren, mit denen *Ether* an andere Accounts gesendet oder empfangen werden kann und sie können Transaktionen an *Smart Contracts* initiieren. Das Erstellen eines *Externally Owned Accounts* kostet nichts.

Contract Accounts *Contract Accounts* werden nur vom Programmcode im *Contract* gesteuert. *Contract Accounts* können Transaktionen lediglich als Reaktion auf erhaltene Transaktionen von einem *Smart Contract* oder *Externally Owned Account* senden. Transaktionen können das Ausführen von Code auslösen, der wiederum verschiedene Aktionen ausführen kann, z. B. das Übertragen von *Token* oder das Erstellen eines neuen *Contracts*. Das Erstellen eines *Contract Accounts* ist mit Kosten verbunden, da Netzwerkspeicher allokiert wird.

Logs *Contracts* können Daten in einer speziell indizierten Datenstruktur, den *Log-Trees*, speichern. In *Solidity* werden *Logs* verwendet um *Events* zu implementieren. Der *Contract* kann nach der Erstellung nicht auf Logdaten zugreifen. Von Clients außerhalb der

Blockchain ist der Zugriff jedoch möglich.[15]

Message Calls Über *message calls* kann weiterer *Contractcode* aufgerufen und ausgeführt oder Transaktionen initiiert werden. So entstehen komplexe *Contracts* mit tief verschachtelten Aufrufen. *Message Calls* sind wie Transaktionen aufgebaut und enthalten den Empfänger und Absender der Nachricht sowie Ether- und Rückgabedaten, jedoch keinen *Gas*-Wert und keine Signatur des Senders.[38] Der Aufrufer, der die Transaktion initiiert, bestimmt den *Gas*-Wert. *Message Calls* werden von der *Ethereum Virtual Machine* initiiert und ausgeführt, daher ist keine Signatur des Senders notwendig.[98] Die *Ethereum Virtual Machine* verfügt über vier Anweisungen für Nachrichtenanrufe: *call*, *callcode*, *delegatecall* und *staticcall*. Die folgende Darstellung zeigt einen Nachrichtenanruf durch eine *call*-Anweisung.

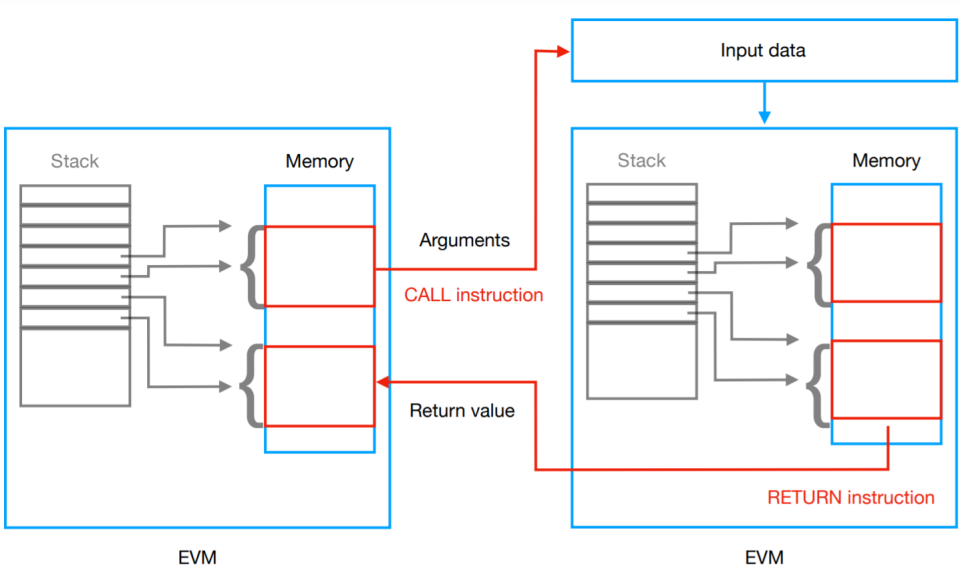


Abbildung 6.4: *Message call* in *Ethereum*, eigene Darstellung nach [34].

Gas Da für jede Transaktionsausführung Rechenressourcen aufgebracht werden müssen, ist für jede Transaktion eine Gebühr erforderlich. *Ethereum* besitzt dafür ein eigenes Gebührensystem.[38] Die Einheit, die den erforderlichen Rechenaufwand misst, um bestimmte Operationen im *Ethereum*-Netzwerk auszuführen, ist *Gas*. *Gas* bezieht sich auf die Gebühr, die erforderlich ist, um eine Transaktion erfolgreich durchzuführen. Bei der Transaktionsausführung muss das *Gaslimit* festgelegt werden. Wenn das Programm die Transaktions-

[15] Beispielsweise aus einer *DApp* über eine API

ausführung nicht erfolgreich abschließen kann, etwa wenn es in einer Endlosschleife steckt, oder das *Gaslimit* zu gering ist, wird die Transaktion abgebrochen und der ursprüngliche Status des *Contracts* wiederhergestellt.[17] Um *out-of-gas Exceptions* zu vermeiden, sollte daher immer das *Gaslimit* ausreichend groß sein.[17] Das *Gas* der Transaktion wird dem *miner* gutgeschrieben, der die Transaktion in einem Block validiert. Der Sender der Transaktion gibt neben dem *Gaslimit* auch den *Gaspreis* an. Der *Gaspreis* definiert die Anzahl an *Wei*, die pro *Gas* gezahlt werden. Dies führt zu einem flexiblen und dynamischen Markt, in dem der Sender der Transaktion beeinflussen kann, wie schnell die Transaktion bestätigt werden soll. Gas, das nach der Transaktion nicht verbraucht wurde, wird an den Sender zurückgezahlt.[38]

Speicherkonzepte Die *Ethereum Virtual Machine* verfügt über drei Speicherbereiche. Im *Storage*, dem permanenten Speicher, über den jedes Konto verfügt, befinden sich alle *Contract*-Zustandsvariablen. Der *Storage* in *Solidity* ist durch eine *Key–value*-Datenbank realisiert, in der Schlüssel und Werte jeweils 32 Byte groß sind. Den *Storage* kann nur der jeweils zugeordnete *Contract* lesen und beschreiben. Temporäre Werte werden in der *Memory* gespeichert. Die *Memory* wird nach externen Funktionsaufrufen gelöscht und ist wesentlich günstiger als der *Storage*. Die Verwendung des *Stack*, in dem lokale Variablen gespeichert werden, ist quasi kostenfrei, kann jedoch in *Solidity* nur eine stark begrenzte Anzahl von Werten enthalten.

Transaktionen Transaktionen in *Ethereum* sind kryptografisch signierte Aufrufe von *Externally Owned Accounts*. Ein *account* initiiert eine Transaktion, um den Status des *Ethereum*-Netzwerks zu aktualisieren und muss diese an alle Knoten des Netzwerks senden, wobei jeder Knoten Transaktionen initiieren darf. Die Validierung der Transaktion wird durch *miner* ausgeführt. Das Transaktionsobjekt muss mit dem *Private Key* des Absenders signiert werden. Nachfolgend wird der Aufbau beschrieben.

- *recipient* - Empfangsadresse der Transaktion
- *signature* - Signatur, die den Absender identifiziert
- *value* - Betrag, der an den Empfänger gesendet wird
- *nonce* - aktuelle Nonce des Senders
- *gasLimit* - maximale Menge an Gas, die die Anzahl der Rechenschritte vorgibt
- *gasPrice* - Gebühr, die der Absender pro Rechenschritt zahlen muss

Ein Transaktionsobjekt muss mit dem *Private Key* des Absenders signiert werden.

```
{
  "id" : 2 ,
  "jsonrpc" : "2.0" ,
  "method" : "account_signTransaction" ,
  "params" : [
    {
     from: "0xez4b65g7hde108f08530af4df5077c2b3d481e5a",
     to: "0xab3d481ee1080e108530a7df50e108530a481e5a",
     gasLimit: "1800",
     gasPrice: "200",
     nonce: "0",
     value: "1500",
    }}
  ]]
}}
```

State Transition Function Aus technischer Sicht kann *Ethereum* als ein *State Transition System* betrachtet werden. Die *State Transition Function* überführt den Ausgangszustand über eine Transaktion in einen Folgezustand.[38] Der Ablauf der *State Transition Function* ist nachfolgend dargestellt.

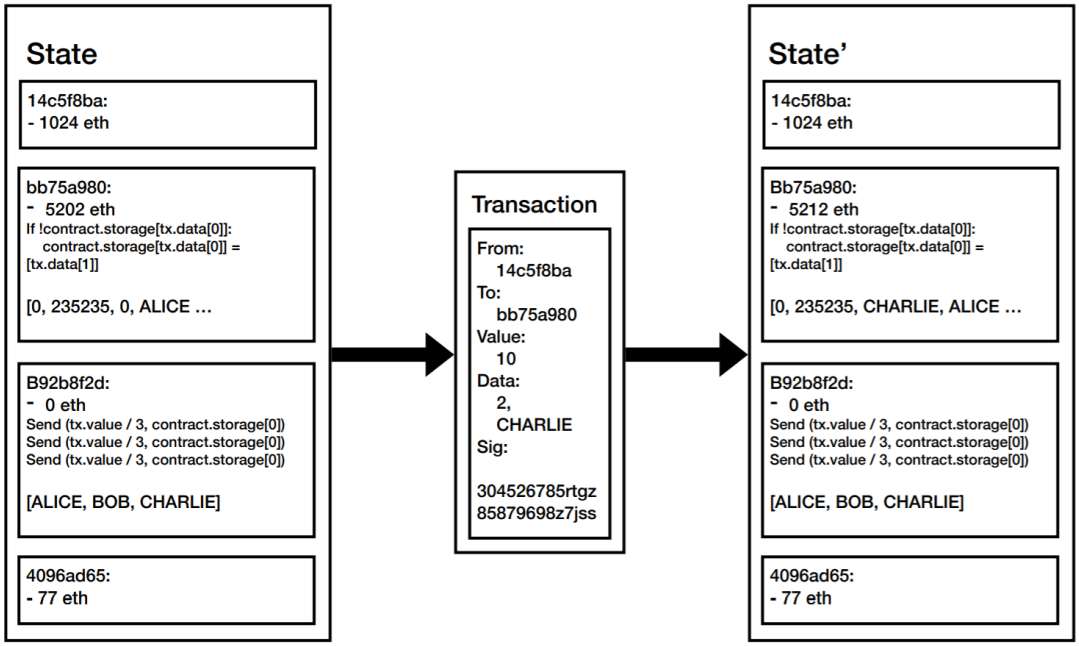


Abbildung 6.5: Zustandsübergangsfunktion in *Ethereum*, eigene Darstellung nach [38].

- Überprüfen der Validität und Struktur der Transaktion, der Gültigkeit der Signatur und des Transaktionszählers sowie des *Nonce*.
- Ermitteln der Senderadresse, Berechnen der Transaktionsgebühr und Abzug der Gebühr vom jeweiligen *account*, Inkrementieren des *Nonce*.
- Übertragen des Transaktionsbetrags und der Daten an den entsprechenden *account*. Ist dieser ein *Contract Account*, wird der *Contract Code* ausgeführt.
- Überweisen des übrigen *Gas* an den Sender. Überweisen der Transaktionsgebühr an den *miner*.

6.3 Solidity

Ethereum Smart Contracts werden in höheren Programmiersprachen geschrieben, die bekannteste und am häufigsten genutzte davon ist *Solidity*, die nachfolgend beschrieben wird. *Solidity* ist eine objektorientierte, anwendungsspezifische höhere Programmiersprache für die Entwicklung von *Smart Contracts*.[113] Die Syntax von *Solidity* weist Ähnlichkeiten zu *JavaScript* auf und wird, im Fall von *Ethereum*[16], vom Compiler zu *Ethereum Virtual Machine Bytecode* kompiliert. *Solidity* ist statisch typisiert, alle Variablentypen müssen zur Kompilierzeit bekannt sein. Sie unterstützt einige Mechanismen der klassischen objektorientierten Programmierung, unter anderem Vererbung, den Einsatz von Bibliotheken und komplexe, benutzerdefinierte Typen.[61]

Zustandsvariablen Zustandsvariablen sind Variablen, deren Werte dauerhaft im *Storage* des *Contracts* gespeichert sind. Sie können gängige Basisdatentypen sein, dazu gehören *Integer*, *Boolean* oder *Byte-Arrays*. *Integer* können sowohl vorzeichenlos als auch vorzeichenbehaftet sein.[63] Außerdem stehen statische und dynamische *Arrays*, *Mappings*, komplexe Datentypen, Adressen und *Enums* zur Verfügung.[63]

Adressen Der Datentyp *address* kommt in zwei Varianten vor: *address* und *address payable*. Der Datentyp integriert Funktionen für die Übertragung von *Ether* wie *send()* und *transfer()*. Da jedes Konto und jeder *Smart Contract* durch eine eindeutige Adresse dargestellt wird, muss in fast jedem *Smart Contract*-Projekt mit Adressen gearbeitet werden. Es wird zwischen *address* und *address payable* unterschieden. Beide beinhalten den *balance* sowie die Funktionen *call()*, *callcode()* und *delegatecall()*. Der Datentyp *address payable* beinhaltet zusätzlich *transfer()* und *send()*. Somit kann an *address payable Ether* gesendet werden, an *address* nicht, was für die Implementierung von *Smart Contracts* im *Smart Grid* von Bedeutung ist. Für Transaktionen im *Smart Grid* sollte die sicherere *transfer()*

[16] *Ethereum* ist nicht die einzige Blockchain-Plattform, die *Solidity* unterstützt

Funktion verwendet werden. Sie benötigt weniger *Gas* und löst im Fehlerfall einen *revert()* aus und gibt zusätzlich eine *Exception* zurück. *send()* gibt lediglich *false* zurück.

Funktionen In *Solidity* gibt es vier Arten von Funktionen: *external*, *internal*, *public* und *private*.

- *external*: Kann über Transaktionen und über andere *Contracts*, allerdings nicht über interne Aufrufe ausgeführt werden.
- *internal*: Kann lediglich über interne Aufrufe ausgeführt werden
- *public*: Kann über interne und externe Aufrufe ausgeführt werden
- *private*: Kann über interne Aufrufe ausgeführt werden

Mit *view* oder *pure* deklarierte Funktionen zeigen an, dass sie den Zustand nicht modifizieren.[57] In *Solidity* können *Fallback*-Funktionen implementiert und dann ausgeführt werden, wenn beim Aufruf des *Contracts* keine bestimmte Funktion angegeben ist. Wie jede Funktion kann die *Fallback*-Funktion ausgeführt werden, solange genügend *Gas* an sie weitergeleitet wird.[70] Sie hat keine Argumente und keinen Rückgabewert. Die Implementierung einer *Fallback*-Funktion ist ein relevanter Sicherheitsmechanismus im Kontext von *Smart Grids*. Auf weitere Sicherheitsmechanismen im Kontext des *Smart Grid* wird in einem späteren Kapitel eingegangen.

Modifier *Modifier* werden verwendet, um das Verhalten einer Funktion zu verändern und deren Funktionalitäten zu ergänzen und, wie in der vorliegenden Implementierung, einzuschränken. *Modifier* sind vererbbare Eigenschaften von Contracts.[57] Funktionen können einen oder mehrere *modifier* enthalten, so können beispielsweise Bedingungen hinzugefügt und deren Eintreten beim Aufruf der Funktion geprüft werden.

```
modifier _onlyOwner {
      require(owner == msg.sender);
      _;
}
```

Im Beispiel wird die Adresse des Eigentümers mit der des Senders verglichen. Wenn der Eigentümer diese Funktion aufruft, wird sie ausgeführt, andernfalls wird eine *Exception* ausgelöst. Der *modifier* kann anschließend jeder Funktion hinzugefügt werden, die nur vom Eigentümer aufgerufen werden können soll.

```
function withdraw(unit amount) _onlyOwner {
        if(checkValue(amount)) {
           value -= amount;
        }
}
```

Application Binary Interface Das *Application Binary Interface* ist die Standardmethode, um mit *Contracts* im *Ethereum*-Ökosystem zu interagieren. Es enthält die zur Verfügung stehenden Funktionen eines *Contracts*. Das Standardformat für das *Application Binary Interface* ist *JSON*.[61] Das *ABI* spezifiziert die *Contract*-zu-*Contract* Interaktionen als auch Interaktionen von außerhalb der Blockchain.

Events Beim Aufruf eines *Events* werden die Argumente im Transaktionslog gespeichert. Der Transaktionslog ist eine spezielle Datenstruktur in der Blockchain. Diese *Logs* sind mit der Adresse des Vertrags verknüpft. Anwendungen können *Events* abonnieren und *Callback*-Funktionen beim Eintritt von *Events* ausführen.[57]

Libraries *Libraries* in *Solidity* sind *Contracts*, die wiederverwendbaren Code enthalten und im Kontext des aufrufenden *Contracts* ausgeführt werden. Sobald eine *Library* in der Blockchain bereitgestellt wurde, können ihre Methoden und Eigenschaften von anderen *Contracts* im *Ethereum*-Netzwerk verwendet werden und die *Library* bekommt eine spezifische Adresse zugewiesen.

6.4 Fazit

In diesem Kapitel wurde insbesondere auf syntaktische Besonderheiten eingegangen, die die Programmiersprache *Solidity* und die *Ethereum*-Plattform von anderen Sprachen und Blockchain-Plattformen unterscheiden. So bietet *Solidity* eine breite Palette an Datentypen, jedoch müssen bei der Entwicklung Speicherkonzepte für die interne Speicherung und Bearbeitung der Daten bedacht werden. Darüber hinaus entwickeln sich die *Ethereum*-Plattform und *Solidity* rasant weiter. Allein seit Dezember 2020 gab es mit *Solidity v.0.7.6*, *Solidity v.0.8.0* und *Solidity v.0.8.1* drei neue Versionen.[104] Dies bedeutet in der Entwicklung den Programmcode ständig aktuell zu halten und neue Sicherheitsmuster zu implementieren. Aus den Grundlagen lassen sich ebenfalls bereits erste Problemstellungen für das Smart Grid, beispielsweise die fehlende Ausführungskontrolle nach der Bereitstellung eines *Contracts*, die potenzielle Gefahr von *callbacks* oder unüberschaubare Kosten bei der unkontrollierten Ausführung von *Contract Account Bytecode* erkennen. Das Kapitel gibt somit eine erste Einführung in die Grundlagen von *Ethereum* und *Solidity*, die insbesondere für das Verständnis der Implementierung von *Smart Contracts* in Kapitel 7

von Bedeutung sind. So wurden die verschiedenen Account-Typen beschrieben und ihre Bedeutung für das *Smart Grid* erklärt.

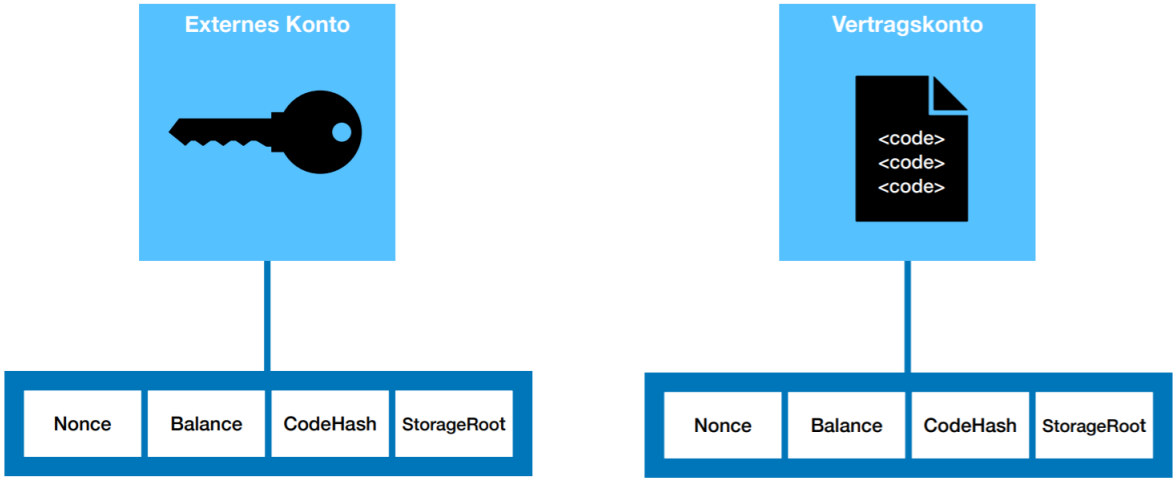


Abbildung 6.6: Die zwei *account*-Typen in *Ethereum*, eigene Darstellung nach [98].

Gleichzeitig dient das Kapitel und die dargestellten potenziellen Gefahren als Grundlage für die Identifizierung von Sicherheitslücken, für die in Kapitel 8 Entwurfsmuster und Problemlösungsstrategien vorgeschlagen werden.

7 Implementierung

7.1 Einleitung

In diesem Kapitel werden prototypisch implementierte *Smart Contracts* für den Aufbau eines *Smart Energy Grid* vorgestellt. Ziel der Implementierung ist es, zu untersuchen, wie *Smart Contracts* in der *Ethereum*-Blockchain eingesetzt werden können, um verschiedene Akteure in einem *Smart Energy Grid* zu agglomerieren und miteinander zu vernetzen. Damit soll ermöglicht werden, kleinste Strommengen dezentral über eine *Peer-to-Peer*-Handelsplattform ohne Intermediäre zwischen einzelnen Akteuren zu übertragen. Zunächst werden dafür in der Implementierung verwendete Tools, *APIs* und *Frameworks* beschrieben. Anschließend werden Möglichkeiten der Zugriffskontrolle auf das *Smart Grid* vorgestellt und die Implementierung drei unterschiedlicher Ansätze beschrieben. Es folgt die Beschreibung eines *Smart Contract* zur Registrierung von *Smart Metern* im *Smart Grid*. Dies ist Voraussetzung, um am Energiehandel teilzunehmen. Anschließend wird die

Implementierung eines *Contracts* vorgestellt, der Kauf- und Verkaufsaufträge registriert, protokolliert und Transaktionen abwickelt.

7.2 Remix IDE

Die *Remix IDE*[17] ist eine Open Source Web- und Desktop-Entwicklungsumgebung. Die *Remix IDE* verfügt über eine Vielzahl von *Plugins* mit einer intuitiven *GUI*. *Remix* ermöglicht die Entwicklung, Bereitstellung, Verwaltung und das Testen von *Solidity Smart Contracts*. *Remix* ist in *JavaScript* geschrieben und unterstützt die Verwendung im Browser und in einer Desktop-Version. Die *IDE* verfügt über eine gute Dokumentation und eine große Entwicklergemeinschaft. *Remix* fördert damit schnelle Entwicklungszyklen, weshalb es als Entwicklungsumgebung für diese Arbeit gewählt wurde.

7.3 ERC20-Token

ERC20 ist ein häufig verwendeter technischer Standard zur Implementierung von *Token* in der *Ethereum*-Blockchain, der für die Implementierung der Kauf-und Verkaufsaufträge im *Contract EnergyTrading.sol* verwendet wurde. Der *ERC20*-Standard integriert Funktionen in den *Smart Contract*, die das Übertragen von *Token* von einem Konto auf ein anderes, die Ausgabe des aktuellen Kontostands oder des Gesamtvorrats der im Netzwerk verfügbaren *Token* ermöglichen. Außerdem ist es möglich, deren Umlauf ständig zu überwachen. Mit dem Standard wird auch die Interoperabilität einzelner *Contracts* erleichtert, da der *ERC20*-Standard ein *API* einrichtet und es *Contracts* ermöglicht, miteinander zu interagieren. Dafür erbt der *Contract* für diese Implementierung vom *OpenZeppelin-Contract StandardToken.sol*.

7.4 Zugriffskontrolle

Im *Smart Grid* müssen verschiedene Akteure auf *Contract*-Funktionen zugreifen und in Echtzeit Informationen austauschen. Eine der zentralen und kritischen Funktionalitäten des *Smart Grid* stellt daher die Zugriffssteuerung dar. Damit kann der Zugriff auf Ressourcen eingeschränkt und Sicherheits- und Datenschutzverletzungen verhindert werden.[27] Native Zugriffskontrollen auf sicherheitskritische Funktionen sind in *Solidity* nicht vorhanden.[60] Die Implementierung eigener Authentifizierungs- und Autorisierungsmechanismen ist daher für jede Anwendung von zentraler Bedeutung. Eine einfache Zugriffssteuerung stellt die Implementierung eines *modifiers* dar, der den aktuellen Besitzer des *Contracts* zurückgibt und Funktionsaufrufe auf den Besitzer beschränkt. Die Implementierung basiert auf der *OpenZeppelin Smart Contract*-Implementierung *Ownable.sol*.[86]

[17] https://remix.ethereum.org/

```
modifier onlyOwner() {
    require(msg.sender == currentOwner, "calling address is not the owner");
    _;
}
```

Der *onlyOwner modifier* löst einen *revert* in einer Funktion aus, die nicht von der als Eigentümer registrierten Adresse aufgerufen wird. Für administrative Zwecke, zum Beispiel dem Zuweisen von Rollen, kann der Mechanismus verwendet werden. Im *Smart Grid* ist er aber auf Grund seiner Beschränkung auf eine Rolle nur bedingt als allgemeine Zugriffssteuerungsmethode geeignet. Für eine differenzierte Zugriffskontrolle ist die Implementierung einer *Allowlist* möglich, die eine aktualisierbare Positivliste genehmigter Adressen führt.

```
contract Allowlist is Ownable {
    mapping(address => bool) allowlist;
    event MemberAdded(address indexed member);
    event MemberRemoved(address indexed member);

constructor() Ownable() {
   }
```

So werden Adressen einzelner Accounts, die *Smart Meter* repräsentieren, auf der Plattform registriert und der *Allowlist* zugefügt. Dadurch wird das Empfangen oder Senden von *Tokens* und der Energiehandel über den *Smart Contract* genehmigt.

```
    function isOnList(address _member) public view returns(bool) {
    function removeMember(address _member) public onlyOwner {
    function addMember(address _member) public onlyOwner {
```

Durch das Erben von *Ownable* und das Aufrufen des Konstruktors wird sichergestellt, dass die Adresse, die den *Contract* bereitstellt, als Eigentümer registriert ist. Der *Contract* stellt Funktionen zum Hinzufügen oder Löschen von Adressen bereit. Nur der Besitzer des *Contracts* kann Adressen zur *Allowlist* hinzufügen. Die Funktion *isOnList()* gibt zurück, ob sich eine übergebene Adresse auf der *Allowlist* befindet. Eine prototypische Implementierung der *Allowlist* ist in der Funktion *sellEnergy()* aufgezeigt. Sie bekommt als Parameter beim Funktionsaufruf die Adresse übergeben und prüft, ob diese Teil der *Allowlist* ist und verarbeitet davon abhängig den Auftrag.
Die *Allowlist* hat wiederum selbst nur einen *Contract Owner*, der Adressen hinzufügen darf. Sie basiert daher auf dem *Contract Ownable.sol*. Weiterer Nachteil ist der kontinuierliche Pflegeaufwand. Eine rollenbasierte Zugriffskontrolle [18] bietet diesbezüglich deutlich mehr Flexibilität. Hier werden mehrere Rollen definiert, die jeweils unterschiedliche Aktionen

[18] Role-Based Access Control

```
function sellEnergy(address member, uint32 value, uint64 amount, uint64
    timestamp)
require(allowlist.isListed(member), "The Account is not listed on the
    Allowlist.");
```

ausführen dürfen und unterschiedliche Berechtigungsstrukturen haben. Separat können Regeln definiert werden, wie Adressen eine Rolle zugewiesen und übertragen werden kann.

```
function createRole(string memory _description, uint256 _admin) public
    returns(uint256) {
function isMember(address _account, uint256 _role) public view returns(bool) {
function deleteMember(address _account, uint256 _role) public {
function addMember(address _account, uint256 _role) public {
```

Mit der Implementierung können neue Rollen zur Laufzeit erstellt werden sowie neue Adressen zu Rollen hinzugefügt werden. Dafür stehen vier Funktionen zur Verfügung. Eine Rolle ist durch ein *Struct* dargestellt, bestehend aus einer Gruppe von Mitgliederadressen und dem *admin*, der *ID* des jeweiligen Administrators. Alle Rollen werden im Array *Role[]* gespeichert. Die Adressen repräsentieren *Smart Meter*, die die jeweiligen Rollen tragen. Die Rollenzugehörigkeit von Adressen kann nur vom jeweiligen *admin* der Rolle verändert werden.

```
struct Role {
    string description;
    uint256 admin;
    mapping (address => bool) members;
}
```

7.5 Registrierung

Um am *Peer-to-Peer*-Energiehandel teilnehmen zu können und Strom zu handeln, müssen *Smart Meter* in der Blockchain registriert werden. Dieser Prozess kann über die *Access Control* eingeschränkt werden. Der *Contract RegisterSmartMeter.sol* implementiert dabei Funktionen der Zugriffssteuerung der *RoleBasedAccessControl*. Die Referenzierung von *Smart Metern* erfolgt über die dezidierte *Ethereum*-Adresse.

Die registrierten *Smart Meter* werden in einem Array gespeichert. Der *Contract* hat zwei *modifier*. Der *modifier onlyMember* prüft intern, ob eine Adresse, die sich im *Smart Grid* registrieren möchte, berechtigt ist, dies zu tun. *OnlyRegisteredSmartMeter* kann in einem weiteren *Contract* eingesetzt werden, um zu prüfen, ob eine Adresse, die Strom handeln möchte, im *Smart Grid* registriert ist.

Die Funktion *registerMeter()* registriert eine Adresse im *Smart Grid*. Der *modifier only-*

```
contract RegisterSmartMeter is RoleBasedAccessControl{

    mapping (address => bool) registeredSmartMeter;
    address[] public registeredSmartMeters;

constructor() public payable {
    addRootRole("GRID_ADMIN");
}
```

```
modifier onlyMember(uint256 roleId) {
    require(isMember(msg.sender, roleId), "Restricted to members.");
    _;
}
modifier onlyRegisteredSmartMeter {
    require (registeredSmartMeter[msg.sender] == true);
    _;
}
function registerMeter() public onlyMember(roleId){
    registeredSmartMeter[msg.sender] = true;
    registeredSmartMeters.push(msg.sender);
}
```

Member prüft, ob *msg.sender* Mitglied der Rolle ist. Dann wird der *Event registeredSmartMeters* angestoßen. Damit wird die *msg.sender*-Adresse übergeben und im Array gespeichert. Nach der Registrierung ist die Adresse des jeweiligen *Smart Meter* nun im Array *registeredSmartMeters* vom Datentyp *address[]* persistiert. Bei Kauf-oder Verkaufsaufträgen kann der *modifier onlyRegisteredSmartMeter* zu einer Funktion hinzugefügt werden, der als erstes die Zugehörigkeit prüft.

7.6 Stromhandel

Der *Contract EnergyTrading.sol* stellt den Kern der Implementierung der *Peer-to-Peer*-Energiehandelsplattform dar. Damit können Prosumer einerseits überschüssige Energie auf der Plattform anbieten und andererseits über Verkaufsaufträge anderer Prosumer Energie beziehen. Über die *Peer-to-Peer*-Energiehandelsplattform entfällt dadurch die Notwendigkeit eines Brokers als Intermediär. Prosumer, die an der Plattform teilnehmen wollen und Strom handeln möchten, müssen sich zuvor über den *Contract RegisterSmartMeter.sol* registrieren.

```
contract EnergyTrading is RegisterSmartMeter, StandardToken {
address public energySeller;

mapping(address => uint) public sellIndex;
mapping(address => uint) public buyIndex;
```

Um den *modifier onlyRegisteredSmartMeter* der Registrierung nutzen zu können, erbt der

Contract von *RegisterSmartMeter.sol*. *StandardToken* integriert den *ERC20-Token* in die Plattform. Registrierte *Smart Meter* können über den *Contract* Kauf- und Verkaufsaufträge erstellen. Die Aufträge im *Smart Grid* sind durch *Structs* abgebildet.

```
struct EnergySellingOrder {
    address seller;
    uint64 price;
    uint64 energyAmount;
}

struct EnergyBuyingOrder {
    address seller;
    uint64 price;
    uint64 energyAmount;
    address meterAddress;
}
```

Die *Structs* bestehen aus der Adresse des Verkäufers, dem Preis, der Strommenge und der *Smart Meter*-Adresse des Käufers. Der *Contract* protokolliert die erstellten Aufträge. Von einem Verkäufer angebotener Strom kann dezidiert über den *sellIndex*, Kaufaufträge über den *buyIndex* referenziert werden.

```
mapping(address => uint) public sellIndex;
mapping(address => uint) public buyIndex;
```

Ein *Smart Meter*, der Strom verkaufen möchte, ruft die Funktion *sellEnergy()* auf. Die Funktion hat zwei Bedingungen: eine Mindestabgabemenge von Strom und einen Mindestpreis. Anschließend wird der erstellte Verkaufsauftrag im *Contract* protokolliert und indiziert, um später über die Funktion *buyEnergy()* aufgerufen werden zu können.

```
function sellEnergy(uint64 aprice, uint64 aenergyAmount )
    onlyRegisteredSmartMeter public {
    require(aprice >= 1);
    require(aenergyAmount >= 1);
    uint recordOrder = sellIndex[msg.sender];
```

Anschließend wird der aktuelle Verkaufsauftrag in das *Struct sellingOrders* geschrieben. Dabei wird der Verkäufer, der Preis und die Energiemenge übergeben. Abschließend wird der *Event energySold* aufgerufen und die Werte damit in der Blockchain gespeichert.
Über den selben *Smart Contract* können auch Kaufaufträge abgewickelt und Strom bezogen werden. Dafür werden dezidierte Verkaufsaufträge aufgerufen. Über den Index wird der spezifische *sellIndex* der Funktion *sellEnergy()* referenziert.
Nun wird in der Funktion *buyEnergy()* zunächst geprüft, ob das entsprechende Angebot existiert. Als nächstes prüft die Funktion, ob der Preis des Käufers mit dem festgelegten

```
sellingOrders.push(EnergySellingOrder({ seller: msg.sender, price: aprice,
    energyAmount: aenergyAmount }));

emit energySold(sellingOrders[recordOrder].seller,
    sellingOrders[recordOrder].price, sellingOrders[recordOrder].energyAmount);
```

```
 function buyEnergy(address aseller, uint64 aprice, uint64 aenergyAmount,
     address mAddress) onlyRegisteredSmartMeter public {
   uint recordOrder = sellIndex[aseller];

   if ((sellingOrders.length > recordOrder) &&
       (sellingOrders[recordOrder].seller == aseller)) {
      require(sellingOrders[recordOrder].price == aprice);
      require(aprice >= 1);
      require(aenergyAmount >= 1);

     buyIndex[msg.sender] = buyingOrders.length;
```

Verkaufspreis übereinstimmt. Nun wird der Kaufauftrag im Index *buyIndex* gespeichert.

```
  buyingOrders.push(EnergyBuyingOrder({ seller: aseller, price: aprice,
      energyAmount: aenergyAmount, meterAddress: mAddress }));
    emit energyBought(aseller, aprice, aenergyAmount, mAddress);
    emit Transfer(0x0, msg.sender, aprice);

    require(buyingOrders[recordOrder].seller == energySeller);
      sellerEnergy.push(EnergyBuyingOrder({ seller: aseller, price: aprice,
          energyAmount: aenergyAmount, meterAddress: mAddress }));
```

Der Kaufauftrag wird in das *Array buyingOrders* geschrieben und der Verkäufer, der Preis und die Strommenge übergeben. Nun wird der *Event energyBought* initiiert und damit die Argumente in der Blockchain persistiert. Der *Event Transfer* aus dem Contract *Standard-Token* initiiert den Übertrag mit *ERC20-Token* im Kontext der ausführenden Adresse.

7.7 Fazit

Mit der prototypischen Implementierung der vorgestellten *Smart Contracts* wurde demonstriert, dass der Einsatz von *Smart Contracts* für den Echtzeit-Energiehandel geeignet ist. Es wurden zunächst verwendete Tools, *APIs* und *Frameworks* beschrieben. Für Anwendungen im Energiesektor sind insbesondere die Implementierung sicherer und transparenter Authentifizierungs- und Autorisierungsmechanismen von zentraler Bedeutung, da verschiedene Akteure auf *Contract*-Funktionen zugreifen und in Echtzeit Informationen austauschen müssen. Es wurden *Smart Contracts* vorgestellt, mit deren Hilfe der Zugriff auf Ressourcen eingeschränkt und eine sichere Zugriffskontrolle umgesetzt werden kann. Dies ist ein wichtiger Teil beim Aufbau eines flexiblen, sicheren und resilienten *Smart Grid*.

Die Implementierung des Registrierungsprozesses von *Smart Meter*-Adressen im *Smart Grid* hat demonstriert, dass eine Energiehandelplattform dezentral und sicher mit *Smart Contracts* umgesetzt werden kann. Damit kann die Teilnahme am Smart Grid perspektivisch selbstverwaltend organisiert werden. Die prototypische Implementierung der *Peer-to-Peer*-Energiehandelsplattform hat gezeigt, dass der Einsatz von *Smart Contracts* für den Echtzeit-Energiehandel geeignet ist. Damit kann eine Balance zwischen Stromerzeugung und Stromverbrauch hergestellt werden. Dies ist der erste Schritt hin zum erzeugungsabhängigen Verbrauch in einem intelligenten Stromnetz und wichtiger Schritt auf dem Weg zu einer intelligenten Sektorenkopplung. Die prototypische Implementierung hat gezeigt, dass in einem Blockchain-basierten Smart Grid Verbraucher aktiv in den Strommarkt einbezogen werden können. Dabei entstehen neue Kommunikationsprozesse, die mit Hilfe von Smart Contracts sicher und gegen Angriffe von außen geschützt realisiert werden können.

8 Sicherheitsanalyse

Die Integration der Blockchain in das *Smart Grid* verbessert die Effizienz und Verfügbarkeit des Stromversorgungssystems. Mit *Smart Contracts* können die Anforderungen der Nutzer überwacht, gesteuert und verwaltet werden. Gleichzeitig ergeben sich durch die Agglomeration der verschiedenen Akteure in einem komplexen *Smart Grid* viele Sicherheitsbedenken und Schwachstellen. In diesem Kapitel erfolgt daher eine Sicherheitsanalyse des Blockchain-basierten *Smart Energy Grid*. Die Analyse betrachtet dabei separat drei Komponenten. Zunächst werden Limitationen sowie potenzielle Hürden beim Einsatz der *Ethereum*-Plattform vorgestellt. Hierauf folgt die Identifizierung häufig aufkommender Angriffsszenarien und Sicherheitslücken in der *Smart Contract*-Entwicklung mit *Solidity*. Anschließend wird der Einsatz der *Ethereum Virtual Machine* als Laufzeitumgebung analysiert und die Vor-und Nachteile beleuchtet. Das Kapitel beschließt mit einer Zusammenfassung der gewonnenen Erkenntnisse.

8.1 Blockchain-Analyse

Im Folgenden werden Herausforderungen beim Einsatz der Blockchain-Technologie allgemein und der *Ethereum*-Plattform im Speziellen aufgezeigt, die in *Smart Grids* mit Elektromobilität Bedeutung haben. Anschließend werden auf Grundlage einer Literatur- und Anwendungsrecherche existierende Lösungsansätze vorgeschlagen.

Datenschutz In *Smart Grids* werden zahlreiche Verbrauchsdaten in Echtzeit erfasst, beispielsweise um Lastprognosen zu erstellen. In einem Blockchain-basierten *Smart Grid* ist es auf Grund der Netzwerkarchitektur der Blockchain wahrscheinlich, dass persistierte

Verbrauchs- und Nutzerdaten unveränderbar für immer in der Blockchain gespeichert werden[19]. Zudem könnten Daten, die mit, nach heutigem Stand der Technik, sicheren kryptographischen Verfahren verschlüsselt werden, perspektivisch mit verhältnismäßig geringem Aufwand entschlüsselt werden. In einem Blockchain-basierten *Smart Energy Grid* sollten nur Daten, die für das Betreiben des *Smart Grid* unbedingt notwendig sind, oder zumindest datenschutzrechtlich unbedenklich, in Klartext in der Blockchain gespeichert werden. Datenschutzrechtlich relevante Daten sollten nur verschlüsselt persistiert werden, beispielsweise in gehashter Form. Ein weiteres Problem stellt die mögliche Analyse von Metadaten dar. So sind in der vorgestellten Implementierung Transaktionen Adressen zugeordnet, die wiederum Rückschlüsse auf personenbezogene Daten zulassen könnten. Um dies zu verhindern, existieren bereits erste technologische Ansätze. Ein möglicher Ansatz, *Assets* und Adressen voneinander zu trennen und so Rückschlüsse auf Metadaten zu unterbinden, ist der Einsatz des *Zero-Knowledge-Proofs*.[5] Im Kontext eines Blockchain-basierten *Smart Energy Grid* muss dieser Ansatz allerdings noch erprobt werden. Insgesamt müssen der Datenschutz und die Datensicherheit in einem Blockchain-basierten *Smart Energy Grid* der Elektromobilität noch weiter analysiert werden. Insbesondere die sichere Kommunikation innerhalb des Smart Grid, sichere Hardware, wie *Wallboxen*, *Smart Meter* und *Smart Meter Gateways*, haben perspektivisch noch viel Forschungsbedarf.

Interoperabilität Die angestrebte Dezentralisierung und Vernetzung einer Vielzahl von Akteuren in *Smart Grids* machen die Sicherstellung der Interoperabilität und Skalierbarkeit der Einzelsysteme zu Schlüsselfaktoren für ein stabiles Energiesystem. Sie sind somit zentral für die Akzeptanz der Technologie und in letzter Konsequenz für das Gelingen der Energiewende. Dabei sind geeignete Kommunikationsstandards für die Systemintegration der Blockchain-Technologie im *Smart Grid* maßgebend. Die Blockchain-Infrastrukturen unterscheiden sich zum Teil jedoch stark voneinander. So entwickeln sich unterschiedliche Systeme, die zwar auf denselben Schlüsseltechnologien basieren, allerdings nicht miteinander kommunizieren können. Für die Schnittstellen und den Datenaustausch müssen Standards definiert werden, damit diese nicht Blockchain-spezifisch kodiert, sondern vielmehr anwendungsübergreifend genutzt werden können. Für die derzeitigen Blockchain-Plattformen gibt es keinen Standard, um die Interoperabilität der verschiedenen Systeme zu gewährleisten und so beispielsweise die Kommunikation und Verbindung von öffentlichen und privaten Blockchains zu gewährleisten. Die *Blockchain Interoperability Alliance* ist ein Zusammenschluss mehrerer Initiativen, die an der Entwicklung eines Industriestandards für die Blockchain-Interoperabilität und eines Internet der Blockchains arbeiten, um ein globales System verbundener Blockchains zu erreichen.[31] Auch gibt es be-

[19] Dies widerspricht dem Recht auf Vergessenwerden des Artikel 17 der DSGVO.[95]

reits konkrete Technologien zur Kommunikation zwischen Blockchains. *Blocknet* ist ein Blockchain-Interoperabilitätsprotokoll, das die Kommunikation, Interaktion und den Austausch zwischen verschiedenen öffentlichen und privaten Blockchains ermöglicht.[35] Es stellt über *Oracles* eine Verbindung zu externen *Off-Chain-APIs* und *Off-Chain*-Diensten und -Anwendungen her und ermöglicht den Austausch beliebiger Daten oder *Smart Contracts*.[35]

Skalierbarkeit Die schlechte Skalierbarkeit gehört zu den größten Problemen Blockchain-basierter Anwendungen im Energiesektor.[114] Durch die zunehmende Anzahl der am *Smart Grid* beteiligten Akteure kommt es zu einem Anstieg an Transaktionen. Auch die Rechenleistung und Bandbreite einzelner *Nodes* stellen dabei einen limitierenden Faktor dar. Mit *Ethereum* sind derzeit maximal 15 Transaktionen pro Sekunde möglich. Eine Lösung für die Skalierbarkeit wurde mit *Ethereum 2.0* bereits vorgestellt.[55] Es setzt bei Transaktionen *Sharding* ein, bei dem das Netzwerk in voneinander getrennte Partitionen, sogenannte *Shards*, aufgeteilt wird.

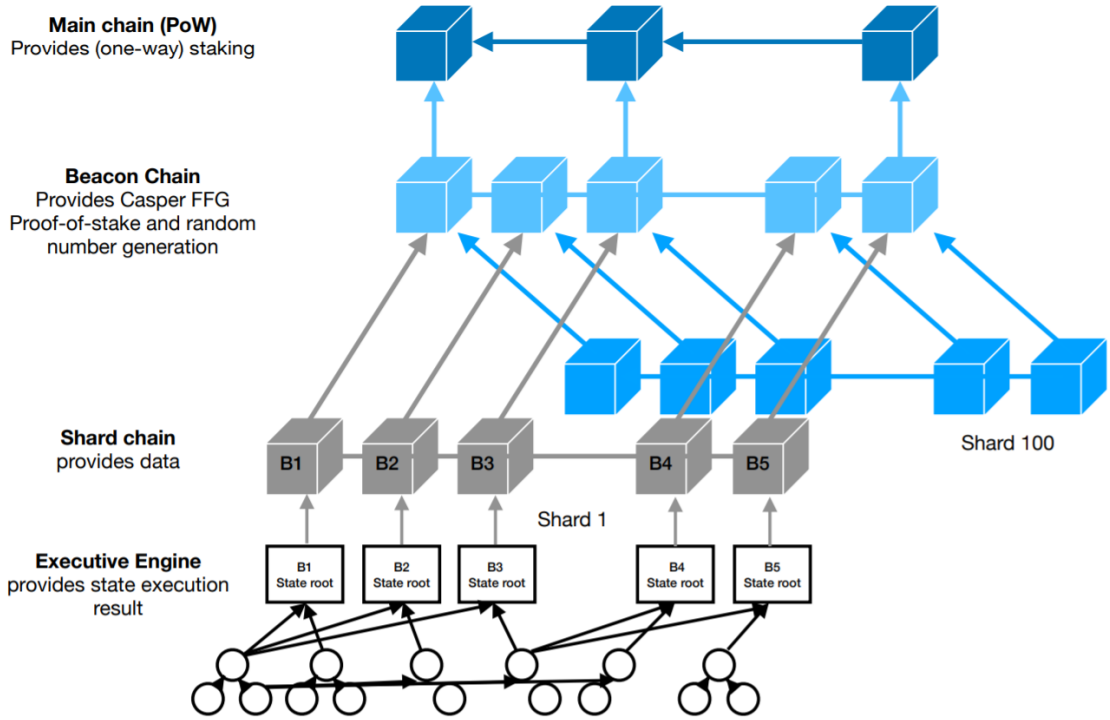


Abbildung 8.1: Skalierung der *Ethereum*-Blockchain durch *Sharding*, eigene Darstellung nach [75].

Diese *Shards* können für bestimmte Funktionen programmiert werden. Die Idee ist, dass für

bestimmte Arten von Transaktionen nur ein *Shard* der Blockchain verwendet werden muss, während die anderen *Shards* für andere Zwecke frei bleiben. Damit soll die Transaktionskapazität erhöht werden. *Ethereum* gibt damit einen 64-mal höheren Durchsatz im Netzwerk an.[55] Weitere *Second Layer*-Lösungen werden derzeit erforscht, um die Skalierungsproblematik anzugehen. So wurden *Off Chain*-[91] und *Side Chain*-Techniken[1] vorgeschlagen, um die Anzahl der Transaktionen zu erhöhen und die Transaktionsvalidierung zu parallelisieren. Ansätze dafür sind der *Distributed Hash-Table*[115], das *InterPlanetary File System*[21] oder *Directed Acyclic Graph-based chains*[25]. Perspektivisch wird der Nutzen der Blockchain-Technologie, speziell in *Smart Grids* des Energiesektors, von technischen Faktoren und Forschungsfortschritten in diesem Bereich abhängen. Bis dahin sollten in einem Blockchain-basierten *Smart Grid* keine großen Datenobjekte gespeichert, sondern nur die wesentlichen Transaktionsinformationen und Hashwerte als Referenz auf Datenobjekte persistiert werden. Eine weitere Lösungsstrategie, die die hohen Latenzen und geringe Skalierbarkeit beheben will, ist der *Unspent Transaction Output*[19], bei dem Transaktionen, die nicht mehr zur Bestimmung von Guthaben benötigt werden, gelöscht werden. Obgleich bereits erste Lösungsansätze existieren, bieten die Skalierungsherausforderungen der Blockchain perspektivisch interessantes Forschungspotenzial. In dieser Arbeit konnte gezeigt werden, dass die Blockchain trotz dieser Problematik die Erwartungen erfüllen und im Kontext des Anwendungsfalls enorme Effizienzsteigerungs- und Kostensenkungspotenziale erbringen kann. *Smart Contracts* können dazu beitragen, Lademanagementstrategien ohne Intermediäre umzusetzen.[89]

Datenintegrität Das Mess- und Eichgesetz der Elektromobilität enthält strenge softwarespezifische Anforderungen bezüglich der Datenintegrität.[52] Es gibt vor, dass relevante Daten vor unzulässigen Veränderungen oder Austausch geschützt werden müssen. Als beschädigt erkannte Daten müssen von der Anwendung verarbeitet werden. Als Nachteil des Einsatzes der Blockchain-Technologie kann hier die Durchführung von Transaktionen in Blöcken und somit in mehreren Datensätzen angesehen werden. Ist eine Transaktion innerhalb eines Blockes der Blockchain nicht valide, kann nicht mehr eindeutig geprüft werden, welcher Datensatz manipuliert wurde. Die Folge ist, dass alle in diesem Block enthaltenen Datensätze geprüft werden müssen. Einen Lösungsansatz kann neben dem Hashen der Blöcke das Hashen der Datensätze selbst liefern.[10] So kann sichergestellt werden, dass manipulierte Daten innerhalb eines Blockes erkannt werden. Gleichzeitig birgt der Ansatz den Nachteil, dass sowohl die benötigte Rechenleistung als auch die Komplexität des Systems ansteigen.

Schnittstellen Bereits bei der Planung müssen entsprechende Maßnahmen zur Überwachung, Auswertung und Abrechnung der Ladevorgänge vorgesehen werden. Die drahtlose Kommunikationsinfrastruktur zwischen Elektrofahrzeugen und dem *Smart Grid*, sowie die Schnittstellen zwischen Elektrofahrzeug, Ladestation und Backend sind potenzielle Schwachstellen. Daher ist deren Sicherung ein entscheidendes Kriterium für eine sichere Nutzung der Ladeinfrastruktur. In der Blockchain persistierte Daten können nur mit erheblichem Aufwand geändert werden Dennoch können Daten schon bei der Übertragung manipuliert werden. Hier muss sichergestellt werden, dass nur befugte Personen Zugriff auf die von ihnen gerade benötigten Daten haben. Neben der vorgeschlagenen Zugriffskontrolle werden Entwurfsmuster vorgestellt, die zur Sicherung von *Smart Contracts* beitragen.

8.2 Smart Contract-Analyse

Die drahtlose Kommunikationsinfrastruktur zwischen Elektrofahrzeugen und dem *Smart Grid* sowie bisher fehlende Standards können zu schwerwiegenden Sicherheitslücken führen, denen begegnet werden muss.[49] Da die Transaktionsausführung in *Smart Contracts* mit Kosten verbunden ist und *Smart Contracts* nach der Implementierung unveränderlich in der Blockchain existieren, haben Fehler eine größere Sicherheitsrelevanz als in klassischen Anwendungen. Fehler ziehen meist direkte finanzielle Konsequenzen nach sich.[92] Daher ist es wichtig, dass *Contracts* weitestgehend frei von Programmierfehlern sind, die zu schwerwiegenden Sicherheitslücken führen können. Potenzielle Gefahren müssen vorher analysiert und erkannt werden. Nachfolgend werden Entwurfsmuster und Problemlösungsstrategien vorgeschlagen.

Transaction Ordering Dependence Jeder Block enthält eine Reihe von Transaktionen, mit deren Ausführung sich *Contract*-Zustände und damit der Zustand der Blockchain verändern können. Bei der *Transaction Ordering Dependence* wird die Reihenfolge, mit der Transaktionen in einen Block aufgenommen werden, manipuliert. *Miner* können, beispielsweise über die Ordnung der Transaktionen, den Zustand der Blockchain beeinflussen.[12] Dieses Verhalten kann bei der Implementierung eines *Smart Contracts* zur Aushandlung von Strompreisen im Grid eine Rolle spielen, bei der Preise häufig aktualisiert werden. So kann beispielsweise der ausgehandelte Kaufbetrag noch während der Verarbeitung der Transaktion geändert werden.

```
contract TransactionOrderingDependence {
  uint64 energyPrice;
  uint64 numerator;

  event UpdateEnergyPrice(address _seller, uint64 _energyPrice);
```

Als Lösung wird die Implementierung eines Transaktionszählers vorgeschlagen. Der Transaktionszähler kann einen Betrag nach einer ausgehandelten Vereinbarung für die Transaktion sperren.[45] Für das Prüfen einer *Transaction Ordering Dependence* im Kontext des ausführenden *Contracts* kann der *numerator* direkt zu Beginn geprüft werden. Gab es eine Änderung, wird ein *revert* geworfen. Alternativ kann der *modifier checkDependence* verwendet werden.

```
modifier checkDependence(uint64 _numerator) {
       require(_numerator == numerator);
       _;
   }
```

Reentrancy Zu den häufigsten Sicherheitslücken bei der Entwicklung von *Smart Contracts* auf der *Ethereum*-Blockchain zählt der Wiedereintritt in Methoden (engl.: *Reentrancy*).[4] *Smart Contracts* können während ihrer Ausführung andere *Contracts* ausführen. Ein Problem entsteht dann, wenn ein *Contract* in einem inkonsistenten Zustand aufgerufen wird und der Wiedereintritt destruktive Änderungen des Zustands bedeutet. *Reentrancy* entsteht dabei meist durch externe Aufrufe von *Smart Contracts*, bevor alle lokalen Änderungen durchgeführt wurden. Ein Beispiel ist der erneute Aufruf einer Funktion zur Auszahlung von Token, bevor der ursprüngliche Methodenaufruf zurückkehrt.[60]

```
function withdrawCredit() public {
    uint creditWithdrawal = userBalances[msg.sender];
    (bool success, ) = msg.sender.call.value(creditWithdrawal);
    require(success);
    userBalances[msg.sender] = 0;
}
```

Hohe Verluste erlitt durch das Ausnutzen dieser Sicherheitslücke die *Decentralized Autonomous Organization* beim *DAO-Exploit*.[3] In dem oben stehenden *Solidity*-Beispiel wird das Guthaben *userBalance* des Benutzers intern erst am Ende der Funktion auf 0 gesetzt. Das Guthaben wird allerdings vorher abgebucht. Der Aufrufer kann also mit *call()* die Funktion wiederholt aufrufen und den Betrag mehrfach abbuchen. Dieser Sicherheitslücke kann beispielsweise mit dem *Checks-Effects-Interactions* Entwurfsmuster begegnet werden.[87] Mit dem Entwurfsmuster wird der Ablauf der Funktionslogik definiert und es wird sichergestellt, dass alle lokalen Änderungen abgeschlossen sind, bevor externe Funktionen aufgerufen werden.[87]
Der Name leitet sich aus dem dreiteiligen Aufbau ab. Zunächst werden die Adressen und Ethermenge überprüft (*Checks*). Danach erfolgt das Ändern von Zustandsvariablen (*Effects*). Abschließend werden die Interaktionen mit anderen *Contracts* ausgeführt. Für den

```
function withdrawCredit() public {
    uint creditWithdrawal = userBalances[msg.sender];
    userBalances[msg.sender] = 0;
    (bool success, ) = msg.sender.transfer.value(creditWithdrawal);
    require(success);
}
```

Einsatz im *Smart Energy Grid* wird die Verwendung der sichereren *transfer()*-Funktion vorgeschlagen, da sie weniger *Gas*[48] benötigt und zudem im Fehlerfall einen *revert()* auslöst.[4] Für einen komplexeren Schutz gegen den Wiedereintritt, der auch funktionsübergreifend angewendet werden kann, wird der Einsatz eines *modifiers*, der als *Mutex* fungiert, vorgeschlagen. Dieser wird im *Contract Reentrency.sol* implementiert.[62]

```
    uint64 private _UNLOCKED = 1;
    uint64 private _LOCKED = 2;
    uint256 private _CONDITION;
```

Es wird eine Variable *CONDITION* deklariert, deren Wert bei Methodenaufruf geprüft wird. Alle Methoden und Statusänderungen werden ausgeführt, solange *CONDITION UNLOCKED* ist. Andernfalls wird der Methodenaufruf unterbunden. Zu Beginn wird *CONDITION* mit *UNLOCKED* im Konstruktor initialisiert.

```
    constructor () {
        _CONDITION = _UNLOCKED;
    }
```

Die eigentliche Verwendung des Entwurfsmusters erfolgt über den *modifier reentrancyProtection*, der den Status prüft und auf Funktionen angewendet werden kann. Durch Verwenden des Entwurfsmusters können mit dem *modifier* versehene Methoden vor mehrfacher, gleichzeitiger Ausführung geschützt werden, um sicherzustellen, dass keine verschachtelten wiedereintretenden Aufrufe vorhanden sind.[62] Die Implementierung basiert auf der *OpenZeppelin*-Bibliothek *ReentrancyGuard*.[86]

Unsichere Zufallsgeneratoren Zudem hat Solidity keine native Methode zur Erzeugung von Zufallszahlen. Komplexe Algorithmen zu deren Berechnung werden schnell teuer. In *Ethereum* sind unsichere Zufallsgeneratoren eine Sicherheitslücke. Das Problem geht aus der Tatsache hervor, dass Transaktionen von mehr als einem Knoten im Netzwerk bestätigt werden müssen. Daher müssen auf jedem Knoten dieselben Zufallszahlen reproduzierbar sein. Damit wird verhindert, dass Zufallswerte vorhersehbar sind oder errechnet werden können.[12] Es wird ein Algorithmus benötigt, der eine Zufallszahl einmal generieren und auf mehreren Knoten verwenden kann. Ein solcher Algorithmus ist der Nutzen

```
modifier reentrancyProtection() {
    require(_CONDITION != _LOCKED);
    _CONDITION = _LOCKED;
    _;
    _CONDITION = _UNLOCKED;
}
```

eines *Random Number Generator*, der zufällige Zahlen außerhalb der Blockchain erzeugt. Möglichkeiten bietet beispielsweise der *DAO Random Number-Contract*, der auf den kryptographischen Hashalgorithmus *SHA3* setzt.[94] Hierbei muss erst der Hash eines Wertes mit einer Kaution hinterlegen werden. Eine weitere Möglichkeit ist *Chainlink VRF (Verifiable Random Function)*.[41]

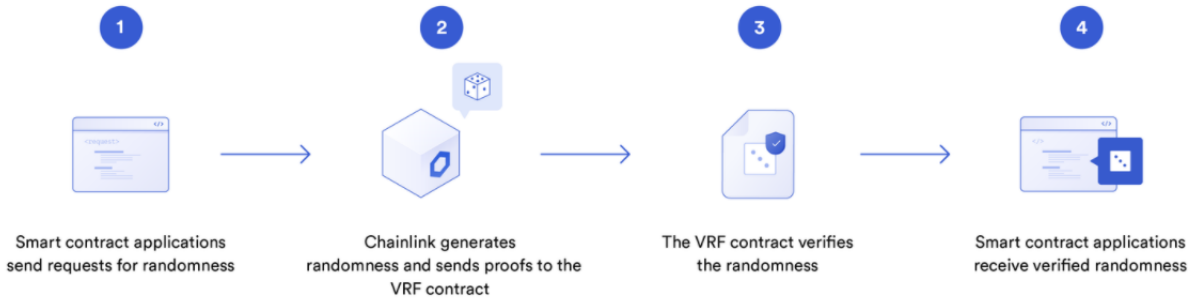


Abbildung 8.2: Funktionsweise von *Chainlink VRF* zur Erzeugung von Zufallszahlen.[41]

Bei jeder neuen Anfrage generiert *Chainlink VRF* eine Zufallszahl und einen kryptografischen Beweis dafür, wie diese Zahl ermittelt wurde. Der *Proof* wird in der Kette veröffentlicht und überprüft, bevor er von Anwendungen verwendet werden kann. Dieser Prozess stellt sicher, dass die Ergebnisse von niemandem manipuliert werden können.[41] Die Bibliothek ist eine schnelle und effektive *Open-Source*-Implementierung von verifizierbaren Zufallsfunktionen in *Solidity*. Die Bibliothek implementiert Verifizierungsfunktionen für *vrf-Proofs* basierend auf der *Elliptic Curve Secp256k1*.

Timestamp Dependence Ein weiteres Sicherheitsproblem ist die Verwendung von Zeitstempeln als Bedingung, um kritische Aktionen anzustoßen. Das Problem hierbei ist, dass Zeitstempel um einige Sekunden von der lokalen Zeit des Systems abweichen und der Block dennoch akzeptiert werden kann. Dadurch kann eine eigentlich vor wenigen Sekunden beendete Auktion noch verändert und so der Zeitstempel des Blocks von *Minern* manipu-

liert werden.[12] Zeitkritische Aktionen sollten demnach nicht auf Zeitstempel, die über *block.timestamp*, *now* oder *block.blockhash* abgerufen werden, vertrauen.[64] Eine mögliche Alternative ist der Einsatz eines *modifiers* und der Blockzeit. Der *modifier* kann eingesetzt werden, um beispielsweise beim Stromhandel zu prüfen, ob der Tarif festgelegt und der Kauf beendet ist, bevor auf bestimmte Funktionen zugegriffen werden kann.[92]

```
modifier auctionComplete {
    require(auctionFinished <= block.number)
_;
}
```

Ungeprüfte Rückgabewerte Eine weitere Sicherheitslücke stellen auch in Solidity ungeprüfte Rückgabewerte dar. Rückgabewerte sind gute Indikatoren für Fehler. Ihre Validierung ist ein wichtiger Aspekt der *Smart Contract* Entwicklung, insbesondere im Umgang mit externen Bibliotheken. Speziell bei Tokentransferfunktionen sind ungeprüfte Rückgabewerte daher problematisch. Mit dem Einsatz von *Exceptions* lassen sich diese oft umgehen. Die *low level* Funktionen *call(), callcode(), delegatecall()* und *send()* lösen jedoch keine aussagekräftigen Exceptions aus. Sie geben lediglich *boolean false* zurück.[43] Nach Möglichkeit sollte die *transfer()*-Funktion verwenden werden, die bei einem Fehlschlagen eine *Exception* auslöst und einen *Rollback* durchführt.[71] Eine weitere Möglichkeit, einen *Rollback* bei Inkonsistenzen auszulösen, bietet die Funktion *require()*, die ebenfalls bevorzugt verwendet werden sollte.

Emergency Stop Auch in gut getestetem Code und zuverlässig funktionierenden *Contracts* können fehlerhafte oder veraltete Codesegmente enthalten sein, insbesondere auf Grund der rasanten Entwicklung von *Ethereum*.[104] Diese werden häufig erst entdeckt, wenn sie von einem Angreifer ausgenutzt werden.[20] Auf Grund der Unveränderlichkeit von *Smart Contracts* und der Blockchain ist es schwer, kritische Sicherheitslücken nachträglich zu beheben. Eine mögliche Problemlösungsstrategie stellt die Implementierung des *Emergency Stop Pattern* dar.[80] Der *Emergency Stop* verhindert die Ausführung aller kritischen Funktionen des *Contracts*, sobald die Erfüllung bestimmter Bedingungen auf Fehler hinweist.

Durch Implementieren des *Emergency Stop Patterns* wird eine zuverlässige Methode integriert, um sensible und kritische Codesegmente im *Contract* zu stoppen, sobald ein Sicherheitsproblem erkannt wird.[14] Dadurch kann Zeit gewonnen und mögliche Sicherheitsverletzungen behoben werden.

[20] Etwa in einer *Zero Day Exploit* Attacke

```
contract StopInEmergency {

    bool emergencyStop = false;

    modifier stopInEmergency {
        require(!emergencyStop);
        _;
    }

    modifier isStopped {
        require(emergencyStop);
        _;
    }
```

Zugriffslimit Mit dem hochfrequenten Energiehandel im *Smart Grid* zwischen Erzeugern und Verbrauchern kommt es zu wiederholten Transaktionen und Funktionsaufrufen innerhalb eines bestimmten Zeitintervalls. In dieser Arbeit wird daher ein Entwurfsmuster vorgeschlagen, mit dem Funktionsaufrufe innerhalb eines Zeitintervalls beschränkt oder nur eine bestimmte Menge Energie über einen Zeitraum gehandelt werden kann.

```
contract RateLimit {
    uint period;
    uint timeLimit;
    uint etherLimit;
    uint lastBlockInPeriod;
    uint withdrawnAmount;
```

Das im Smart Contract *RateLimit.sol* implementierte Entwurfsmuster regelt, wie oft ein Funktionsaufruf in einem Zeitintervall aufrufbar ist und wieviel *Ether* transferiert werden darf. Dafür werden *modifier* bereitgestellt, die das Gaslimit und Zeitlimit als Parameter übergeben bekommen.

```
modifier timeLimiter(uint time) {
    require(block.timestamp >= timeLimit, "Request could not be processed.
        Please try again later.");
    timeLimit = block.timestamp + time;
    _;
}
```

Eine prototypische Implementierung des Zugriffslimits ist mit der *withdraw()*-Funktion im *Contract RateLimit.sol* gegeben. Dieser nutzt den *modifier timeLimiter()* und ruft zu Beginn die *updateRateLimit()*-Funktion auf. Am Ende der Prüfungen wird die *transfer()*-Funktion aufgerufen.

Denial of Service *Distributed Denial-of-Service*-Attacken haben sich zu einer der kritischsten Bedrohungen dezentraler Anwendungen entwickelt. Mit einer *DDos*-Attacke ver-

```
    function withdraw(uint amount) public timeLimiter(1 minutes) {

        updateRateLimit();
        uint maximum = withdrawnAmount + amount;
        require(maximum >= withdrawnAmount);
        require(withdrawnAmount + amount < etherLimit);

        withdrawnAmount += amount;
        msg.sender.transfer(amount);
    }
```

suchen Angreifer, Anwendungen zu unterwandern, indem sie den Server mit gezielten Anfragen aus unterschiedlichen Netzwerken überfluten. Auf Grund der Varianz der Angriffsquellen ist es kaum möglich, einen Angreifer zu blockieren, ohne die Netzwerkkommunikation im Ganzen einzustellen. Auch in der *Ethereum*-Blockchain zählen *Denial-of-Service*-Attacken zu den häufigsten Angriffen. So werden große Mengen kleiner oder ungültiger Transaktionen über das Netzwerk gesendet, um einen hohen Datenverkehr zu produzieren und einzelne Knoten zu überlasten. So wird verhindert, dass legitime Transaktionen verarbeitet werden können.[42] Auch durch das gezielte verursachen von *Out-of-Gas-Exceptions* können *DDoS*-Angriffe entstehen.[42] Wenn keine Sicherheitsvorkehrungen getroffen werden, kann eine *DDoS*-Attacke durch das Auslösen eines einfachen *Rollbacks* durch einen Empfänger von *Tokens* vollzogen werden. Ein Angreifer kann so auch bei fehlerhafter Zugriffskontrolle Rechte erlangen und den *Contract* blockieren.[71] Als Lösung wird die Verwendung des *Pull-over-Push*-Entwurfsmusters für externe Aufrufe vorgeschlagen, das alle externen Anrufe voneinander und von der Vertragslogik isoliert.[110]

```
contract PullOverPush {

    mapping(address => uint) amount;

    constructor(){
}
```

Damit verschiebt sich das mit der Übertragung von *Ether* verbundene Risiko auf den Benutzer. Hierbei sollte jeder externe Aufruf in einer eigenen Transaktion isoliert werden, die vom Empfänger des Aufrufs initiiert werden kann. Außerdem sollte das Iterieren über große Arrays vermieden werden, da hier das Fehlschlagen einer einzelnen Transaktion dazu führen kann, dass alle Transaktionen rückgängig gemacht werden. So kann die Ausführung des *Contracts* dauerhaft fehlschlagen.[48] Mit dem *mapping* wird gleichzeitig die Menge an *Ether* festgelegt, die jede Adresse abziehen darf.
Die Erlaubnis erfolgt nun in der *pullPermission()*-Funktion. Diese sollte anstelle einer aktiven *Push*-Transaktion verwendet werden. Anstelle von *<address>.transfer(amount)* wird

```
function pullPermission(address buyer, uint price) public {
    amount[buyer] += price;
}

function withdrawAmount() public {
    uint price = amount[msg.sender];

    require(address(this).balance >= price);
    require(price != 0);

    amount[msg.sender] = 0;

    msg.sender.transfer(price);
}
```

pullPermission(<address>, amount) verwendet. Da *pullPermission() public* ist, kann sie auch von außerhalb des *Smart Contracts* verwendet werden. Für externe Transaktionsaufrufe sollte unbedingt eine sichere Zugriffsbeschränkung implementiert werden.

8.3 Analyse der Laufzeitumgebung

Ethereum Smart Contracts werden von der *Ethereum Virtual Machine* in *EVM-Bytecode* kompiliert. Die *Ethereum Virtual Machine* ermöglicht als Turing-vollständige Maschine die Entwicklung komplexer Blockchain-basierter Anwendungen im *Smart Grid*. Auf diese Weise können, neben Transaktionen und dem Handel von Strom, auch weitere Anwendungsfälle, beispielsweise die Nutzerauthentifizierung und -verwaltung oder Trackinganwendungen im *Smart Grid* umgesetzt werden. Gleichzeitig birgt der Einsatz der *Ethereum Virtual Machine* im Kontext des *Smart Grid* auch einige Nachteile. So birgt der native Datentyp der *Ethereum Virtual Machine*, eine 256 *Bit-Integer*, erste Probleme bezüglich der Performanz. Da Berechnungen, die kleiner als 256 *Bit* sind, zuerst in ein 256 *Bit*-Format konvertiert werden müssen, bevor die *Ethereum Virtual Machine* sie verarbeiten kann, sind die Operationen entsprechend langsam und aufwändig und in letzter Konsequenz auch bei einfacher Verwendung sehr teuer. Ebenfalls unterstützt die *Ethereum Virtual Machine* nativ keine Gleitkommazahlen.[63] Diese können zwar über Bibliotheken eingebunden und deklariert, in *Smart Contracts* aber nicht zugewiesen oder für Berechnungen verwendet werden. Insbesondere für *Token*-basierte Abrechnungsprozesse im *Smart Grid* stellt dies einen Nachteil dar. Eine weitere Beeinträchtigung stellt die Fragmentierung des Speichers dar, da es keine integrierten Mechanismen zur Freigabe allokierten Speichers während der Programmausführung gibt.[58] Wird Speicherplatz benötigt, wird dieser stets neu allokiert. Gleichzeitig gibt es keine Operationen für das Speichermanagement, wie es in anderen Sprachen der Fall ist. Dadurch können hohe Speicherkosten entstehen. Um einige dieser Probleme lösen zu können, wird derzeit die *Ethereum WebAssembly* entwickelt, die eine Alternative

zur *Ethereum Virtual Machine* darstellt und diese perspektivisch ersetzen soll.[59] *Ethereum WebAssembly* ist eine vorgeschlagene Neugestaltung des *Ethereum Smart Contract Execution Layer* unter Verwendung einer deterministischen Teilmenge des *WebAssembly Bytecodes*.[59] Die *Ethereum WebAssembly* soll die Ausführungs- und Ladezeiten optimieren, indem es Programmcode durch vordefinierte Befehle im Vergleich zur schwerfälligen *Ethereum Virtual Machine* schneller konvertieren und ausführen kann.[51] Zudem soll die Möglichkeit integriert werden, *Smart Contracts* in traditionellen Programmiersprachen, wie *C*, *C++* oder *Rust*, zu entwickeln und der Zugriff auf die *WebAssembly Toolchain* ermöglicht werden. Somit stellt die Umstellung der *Ethereum*-Laufzeitumgebung eine vielversprechende Optimierungsmöglichkeit in Bezug auf Ausführungs- und Ladezeiten sowie im Hinblick auf die Implementierung und Ausführung von *Smart Contracts* in *Ethereum* dar. Ihr tatsächlicher Nutzen für den Anwendungsfall bleibt jedoch abzuwarten.

8.4 Fazit

Die Digitalisierung und Dezentralisierung in der Stromversorgung erhöhen die Komplexität des Stromversorgungssystems und machen es damit anfälliger für Angriffe. Im *Smart Energy Grid* werden über *Smart Meter Gateways* enorme Datenmengen ausgetauscht, die auch sensible und schützenswerte Daten enthalten. Die Vernetzung und Koordination der einzelnen Komponenten und der daraus resultierende Kommunikationsbedarf stellen weitere potenzielle Angriffspunkte dar. In einer solch kritischen Infrastruktur müssen potenzielle Sicherheitslücken identifiziert und ausreichende Problemlösungsstrategien implementiert werden. Damit können die Risiken gesenkt und Gefahren abgemildert werden. In diesem Kapitel wurden anhand einer Literatur- und Anwendungsrecherche sowie auf Basis der Implementierung in Kapitel 7 häufig auftretende Sicherheitslücken der *Ethereum*-Plattform sowie von *Solidity Smart Contracts* und der *Ethereum Virtual Machine* identifiziert und dargestellt. Das Kapitel stellt Sicherheitsmuster und Lösungsstrategien vor. Diese helfen, *Smart Contracts* unter sicherheitsspezifischen Aspekten zu bewerten und sicher zu implementieren. Mit den vorgestellten Entwurfsmustern und der Berücksichtigung der Optimierungsmöglichkeiten während der Planung des Anwendungsfalls kann ein sicheres, robustes und belastbares Blockchain-basiertes *Smart Energy Grid* umgesetzt werden. Da sich die Analyse an der Implementierung orientiert, steht die Programmiersprache *Solidity* im Fokus. Einige der vorgeschlagenen Entwurfsmuster lassen sich auch auf andere Programmiersprachen übertragen, wie beispielsweise das *Pull-Over-Push* oder das *Checks-Effects-Interactions*-Entwurfsmuster. Neben sprachspezifischen Sicherheitslücken können auch in der *Smart Contract*-Entwicklung Denk- und Programmierfehler auftreten. Da einmal entwickelte *Smart Contracts* weitestgehend unveränderlich sind, sollte in der *Smart Contract*-Entwicklung vermehrt auf ausreichende Tests vor dem Deployment geachtet wer-

den. Hierbei kann neben Testen durch Codeanalysen auf das Testen in Testnetzwerken zurückgegriffen werden. Beispiele sind die *Remix IDE*, *Ganache* oder *Truffle*. Sprachspezifische Sicherheitslücken treten auch in anderen Sprachen auf, insbesondere wenn sie auf der *Ethereum Virtual Machine* basieren und von deren Vor- und Nachteilen beeinflusst werden. Die Nachteile der *Ethereum Virtual Machine* wurden in diesem Kapitel dezidiert für das *Smart Grid* aufgezeigt. Die 32 *Byte* verlangsamen die Programmiersprachen in ihrer Performanz und verursachen große *Bytecodes*, woraus wiederum hohe Kosten für Berechnungen auf der *EVM* entstehen. Dies macht den geplanten Umstieg der Laufzeitumgebung in *Ethereum* auf *eWASM* sehr interessant, gerade im Kontext des *Smart Grid*. Mit Effizienzsteigerungen durch kleinere Datentypen und einem performanteren Konsensmechanismus entstehen auch Kostensenkungspotenziale, die zu einer Zunahme von Anwendungen im Energiesektor führen kann. Auf der *eWASM*-Laufzeitumgebung sollen außerdem etablierte Programmiersprachen und Bibliotheken genutzt werden können. Entwickler können so auf einen breiteren Erfahrungsschatz zurückgreifen. Außerdem sind die Gefahren bekannter Programmiersprachen meist gut dokumentiert. Diese Analyse hat gezeigt, dass es einige Sicherheitslücken und potenzielle Gefahren in der *Ethereum*-Blockchain gibt, denen jedoch mit den vorgeschlagenen Lösungsstrategien begegnet werden kann. Der Einsatz von *eWASM* als *EVM*-Ersatz ist vielversprechend, jedoch bleibt abzuwarten, inwieweit echte Verbesserungen eintreten werden.

9 Schluss

In diesem Kapitel erfolgt eine abschließende Betrachtung der Arbeit. Dabei werden zunächst die Ergebnisse der Arbeit zusammengefasst und in den wissenschaftlichen Kontext eingeordnet. Anschließend werden Limitationen der Arbeit aufgezeigt. Abschließend wird in einem Ausblick das Potenzial und der Nutzen der Applikation für die Zukunft eingeordnet.

9.1 Zusammenfassung

Um die Forschungsergebnisse dieser Arbeit nachvollziehbar darzustellen, orientiert sich die Ergebnispräsentation an den *Design-Science Research Guidelines* von *Österle et al.*[18], die bei der Präsentation der Forschungsergebnisse eine Trennung in vier Schritten vorschlägt: Analyse, Entwurf, Evaluation und Diffusion.

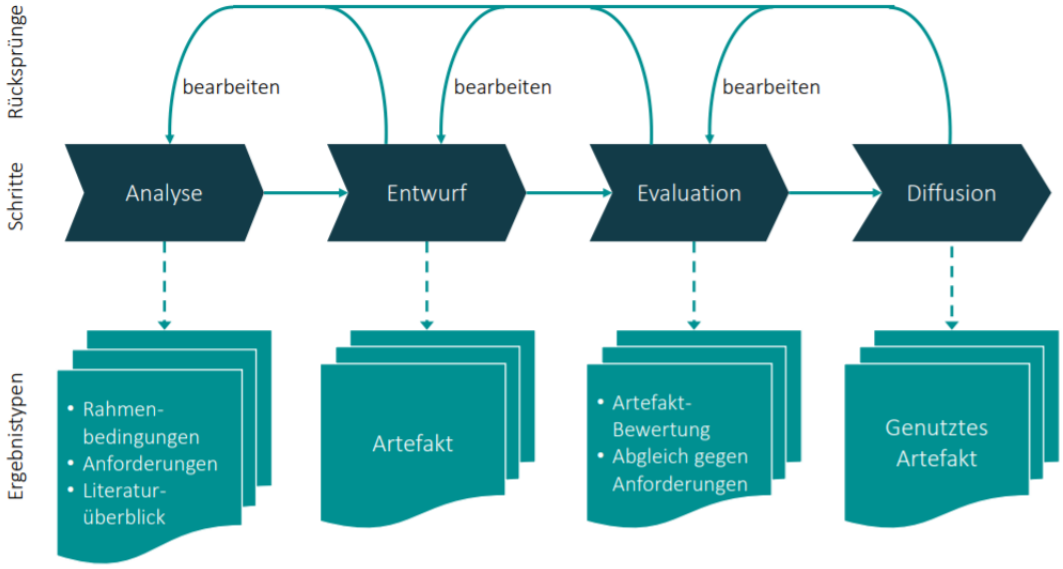


Abbildung 9.1: *Design-Science Research Guidelines* nach Österle et al.[18]

Analyse Das Ziel dieser Arbeit ist es, zu untersuchen, wie in einem *Smart Energy Grid* der Elektromobilität mit Hilfe von *Ethereum Smart Contracts* eine Blockchain-basierte *Peer-to-Peer*-Handelsplattform umgesetzt werden kann. Damit soll der steigenden Komplexität des Stromnetzes mit Flexibilisierungsmaßnahmen begegnet, der Abruf von Regelenergie reduziert und in letzter Konsequenz durch intelligentes Lastmanagement erneuerbare Energien besser integriert werden. Die Arbeit ging der Frage nach, wie der Einsatz der Blockchain-Technologie und *Smart Contracts* die Dezentralisierung und Demokratisierung des Energiesystems begünstigen kann. Hierfür wurde mit der Beschreibung der energietechnischen Grundlagen in den theoretischen Kontext der Arbeit eingeführt und die Schlüsseltechnologien und -bausteine der Blockchain konsistent und vollständig herausgearbeitet. Um eine breite Wissensbasis zu fundieren, wurden in einem Literatur- und Anwendungsüberblick verwandte Arbeiten analysiert. Die Analyse hat gezeigt, dass die wenigen Arbeiten, die sich mit dem Blockchain-Einsatz in *Smart Grids* beschäftigen, den Fokus auf den Aufbau einer stationären *Peer-to-Peer*-Stromhandelsplattform, den Energiehandel zwischen *Prosumern* und Energiedienstleistern oder den Blockchain-basierten Energiehandel zwischen einzelnen *Prosumern* in einer *Community* setzen. Die Erarbeitung des Forschungsstandes hat als Forschungslücke den *Smart Contract* basierten *Peer-to-Peer*-Energiehandel ohne Intermediäre zwischen rückspeisefähigen Elektrofahrzeugen auf der *Ethereum*-Plattform gezeigt und begründet damit das Alleinstellungsmerkmal der Arbeit. Auf der Analyse aufbauend wurden anschließend die Zielgruppen und Stakeholder identifiziert und deren Interessen für den Anwendungsfall dargelegt. In der Analyse haben sich die Herausforderungen dieser Arbeit gezeigt. So entstehen im *Smart Grid* durch den Daten-

austausch sowie durch die Bereitstellung von Echtzeitinformationen enorme Datenmengen. Gleichzeitig herrschen hohe Sicherheitsstandards sowie hohe Ansprüche an die Performanz des Energiesystems. Die Analyse hat gezeigt, dass Lösungen für diese Herausforderungen in den Fokus der Arbeit rücken müssen, um die Vorteile und den Mehrwert des Ansatzes darlegen zu können und die Marktdurchdringung der Blockchain im Energiesektor zu fördern.

Entwurf Auf Basis der Analyse wurden im weiteren Verlauf der Arbeit Merkmale des Blockchain-Designs identifiziert und Interdependenzen zwischen den Merkmalen aufgezeigt. Darauf begründet sich anschließend die Plattformentscheidung sowie die Auswahl geeigneter *Smart Contract*-Programmiersprachen, Konsensmechanismen, der Programmierumgebung und kryptographischer Verfahren. So wurde die Eignung der *Ethereum*-Blockchain und der Programmiersprache *Solidity* begründet. Es wurden Charakteristiken der Plattform sowie syntaktische Besonderheiten der Programmiersprache aufgezeigt. Diese bilden die Grundlage für die Implementierung von *Smart Contracts* in einem *Smart Energy Grid*.

Evaluation In der abschließenden summativen Evaluation werden die Ergebnisse systematisch gegen die identifizierten Anforderungen und Problemstellungen abgeglichen.[18] Die Markt- und Anwendungsanalyse konnte die prinzipielle Wirtschaftlichkeit des Blockchain-basierten *Demand-Side-Management* im *Smart Grid* belegen. Es wurde gezeigt, dass der Einsatz der Blockchain-Technologie allgemein und *Smart Contracts* im Speziellen, die Resilienz und die Flexibilität im Energiemanagement sowie beim Stromhandel erhöhen. Der Handel von Strom unter den Prosumern im Smart Grid und nicht mehr nur mit konventionellen Energielieferanten bewirkt eine sowohl räumliche als auch organisatorische Dezentralisierung des Energiesystems. Aus der dezentralen, verteilten Systemarchitektur der Blockchain und der zunehmenden Dezentralisierung der Energiewirtschaft sind Synergieeffekte zu erwarten, die eine Sektorenkopplung vorantreiben können. Die prototypische Implementierung von *Smart Contracts* auf der *Ethereum*-Blockchain hat gezeigt, dass ein dezentrales Smart Energy Grid der Elektromobilität mit Hilfe der Blockchain-Technologie aufgebaut werden kann, um Strom zu handeln und Transaktionen im Netzwerk konsensfähig ohne Intermediär abzuwickeln. Mit der Implementierung wurden zum einen Lösungen für Standardprobleme aufgezeigt. So wurde beispielsweise die Zugriffskontrolle auf das *Smart Grid* umgesetzt und verschiedene Authentifizierungs- und Autorisierungsmechanismen vorgeschlagen. Zum anderen wurden auch spezifische Lösungen vorgeschlagen und gezeigt, wie mit *Smart Contracts* die Registrierung in der Blockchain umgesetzt werden kann, um an der Energiehandelsplattform teilzunehmen. Daraufhin können Nutzer über *Smart Contracts* Kauf- oder Verkaufaufträge in der Blockchain hinterlegen. Aufträge wer-

den ebenfalls in der Blockchain protokolliert. Auch konnte gezeigt werden, dass *Smart Contracts* für die Transaktionsabwicklung eingesetzt werden und die Übertragung von *Token* in der Blockchain automatisiert umgesetzt werden kann. Die Blockchain-Technologie ist somit eine ideale Basis für die Umsetzung des Anwendungsfalls in einem dezentralisierten Strommarkt mit eingeschränkter Vertrauenswürdigkeit. Die Transaktionshistorie wird dabei in der Blockchain persistiert und auf allen Knoten verteilt. Damit bietet die Blockchain einen zentralen Vorteil gegenüber konventionellen Ansätzen: Sie ist unveränderlich und Transaktionsdetails sind rückverfolgbar. Daraus ergibt sich eine hohe IT- und Cybersicherheit für neue Geschäftsmodelle im Vergleich zu einem konventionellen, zentralisierten Energiemanagement. Die Analyse konnte weitere Herausforderungen identifizieren. Das entstehen enormer Datenmengen, die schlechte Skalierbarkeit der Blockchain sowie die hohen Sicherheitsstandards und hohe Ansprüche an die Performanz des Energiesystems. Für diese Herausforderungen wurden Problemlösungsstrategien aufgezeigt. Es wurden Mechanismen für die bessere Skalierbarkeit von Blockchains und die Sicherstellung der Interoperabilität dargelegt. Aus der Implementierung haben sich zudem Sicherheitslücken der *Smart Contract*-Programmierung mit *Solidity* gezeigt. Diese konnten in einem folgenden Kapitel mit entsprechenden Problemlösungsstrategien behandelt werden. Das in dieser Arbeit entworfene Konzept einer Nutzerapplikation hat gezeigt, dass diese als Schnittstelle zum *Smart Grid* für die verschiedenen Stakeholder klare Vorteile bringen kann, beispielsweise um Informationen über den täglichen Energieverbrauch einzusehen und die damit verbundenen Kosten in Echtzeit abzufragen, Ladevorgänge zu planen, Zahlungen zu initiieren oder Prämien zu überblicken. Die Analyse hat gezeigt, dass der Beitrag eines einzelnen Elektrofahrzeugs zur energiewirtschaftlichen Einbindung stark abhängig von der Fahrzeugnutzung und den freigegebenen Ladekapazitäten ist. Ein Anreizsystem kann die Bereitschaft zur Freigabe eines Teils der Batteriekapazität steigern.

Diffusion Mit der Diffusion gehen die *Design-Science Research Guidelines* nach Österle et al. der Frage nach, wie die Forschungsergebnisse der Arbeit in der Fachwelt verbreitet und dauerhaft nutzbar gemacht werden können. Dies wird mit der Veröffentlichung der vorliegenden Arbeit erreicht.

9.2 Limitation

Mit der Implementierung von *Solidity Smart Contracts* auf der *Ethereum*-Blockchain konnte die technische Machbarkeit des Blockchain-basierten Ansatzes zum Aufbau einer *Peer-to-Peer*-Energiehandelsplattform der Elektromobilität gezeigt werden. Gleichwohl ergeben sich aus der Verwendung der *Ethereum*-Blockchain und der *Solidity*-Programmiersprache Limitationen für den Anwendungsfall. Eine Limitation stellt der Transaktionsdurchsatz

der *Ethereum*-Blockchain dar, die derzeit nur bis zu fünfzehn Transaktionen pro Sekunde verarbeiten kann.[38] Da dies im Protokoll festgelegt ist, muss diese Einschränkung bei der Umsetzung eines *Smart Grid* berücksichtigt werden. *Ethereum* ist zudem eine öffentliche Blockchain: Transaktionen, Nutzerdaten und Adressen sind öffentlich einsehbar. In der vorliegenden Implementierung sind Adressen mit Smart Metern verknüpft, die Rückschlüsse auf deren Verbrauch und Energieaustausch zulassen und damit eine Sicherheitslücke darstellen. In der öffentlichen Blockchain basiert der Preis auf dem Prinzip von Angebot und Nachfrage und ist hochvolatil. So erreichte der *Ether* am 12. Februar 2021 einen neuen Höchststand mit 1835 USD, am 12. Dezember war er lediglich 567 USD wert. Denkbar für den Einsatz im *Smart Grid* ist, dass der *Token*-Preis perspektivisch an eine Fiat-Währung gebunden wird. Eine weitere Limitation dieser Arbeit stellt der nicht vorhandene Zugriff auf reale Daten dar. Der derzeitige technische Stand durch den Einsatz von *Wallboxen* erlaubt, dass aktuelle Verbrauchsdaten zeitkontinuierlich festgehalten und übertragen werden. Um die Erhebung solcher Daten zu ermöglichen, wäre die direkte Verbindung mit einem *Smart Meter Gateway* erstrebenswert gewesen, der die Daten direkt an die Blockchain überträgt, von wo aus über *Smart Contracts* auf sie zugegriffen werden kann. Eine weitere Begrenzung dieser Arbeit liegt in der noch nicht möglichen Würdigung des energetischen und rohstofflichen Aufwands sowie damit verbundener Umweltwirkungen, im Vergleich zum erreichbaren Nutzen. Sogenannte Ökobilanzen liegen zu Blockchains noch nicht vor.[7]

9.3 Ausblick

Dem Einsatz der Blockchain-Technologie im Energiesektor wird oft disruptives Potenzial zugesprochen und die Erwartungen waren und sind groß. Dennoch müssen zunächst grundlegende Limitationen der Blockchain behoben werden. Auch die energiepolitischen Rahmenbedingungen, etwa für den Einsatz von *Smart Contracts*, müssen geschaffen werden, bevor die Blockchain für die Entwicklung neuer Geschäftsmodelle im Energiesektor in der Breite eingesetzt werden wird.[97] Sonst entstehen möglicherweise Rechtsunsicherheiten, die Unternehmen von der Integration Blockchain-basierter Anwendungen abhalten. Ob die Transaktionsabwicklung des *Smart Contract*-basierten Ansatzes teurer ist, als derzeitige zentralisierte Ansätze, hängt unter anderem stark vom Energiebedarf der Blockchain ab. Bei *Proof-of-Work*-basierten Blockchain-Systemen ist dieser sehr hoch. Das gesamte *Ethereum*-Netzwerk verbraucht etwa 19.49 TWh pro Jahr.[21] Für eine einzelne Transaktion beträgt der Strombedarf 42.47 kWh.[22] *Ethereum* plant, den *Proof-of-Work*-Algorithmus durch einen energieeffizienteren *Proof-of-Stake*-Algorithmus abzulösen, der den Energieverbrauch drastisch reduzieren soll.[39] Die tatsächlichen Einsparungen der *Ethereum*-

[21] Entspricht etwa dem Energiebedarf von Island[47]
[22] Entspricht etwa dem durchschnittlichen Stromverbrauch eines Haushalts über 1,44 Tage[47]

Weiterentwicklung bleiben jedoch abzuwarten. Elektrofahrzeuge tauschen permanent Daten aus, ob beim Ladevorgang oder bei der Fahrt, aber auch die Ladeinfrastruktur und Backend-Systeme sorgen für ein enormes Datenaufkommen und permanenten Kommunikationsbedarf der im *Smart Grid* agglomerierten Akteure. Dadurch steht die Interoperabilität verschiedener Systeme im Fokus der Forschung. Ein weiterer zentraler Punkt ist die Cybersicherheit im *Smart Grid*. Mit der Implementierung von Entwurfsmustern oder der Zugriffskontrolle wurden erste Konzepte vorgeschlagen, die die Sicherheit in *Smart Grids* verbessern. Dies ersetzt jedoch nicht eine ganzheitliche Analyse von Datensicherheits- und Datenschutzkonzepten. Die Analyse häufig auftretender Sicherheitslücken sollte in einem nächsten Ausbauschritt um verschiedene Programmiersprachen für *Smart Contracts* erweitert werden, um vergleichend fundierte Schlussfolgerungen der sprachspezifischen Sicherheitsmuster zuzulassen.

Die Blockchain birgt enormes Potenzial für den Energiesektor. In dieser Arbeit wurden Ansätze präsentiert, mit denen sich die Interoperabilität, die Zuverlässigkeit, die Skalierbarkeit und der Energieverbrauch der Blockchain in *Smart Grids* verbessern lassen. Es bleibt abzuwarten, ob sich dieses Potenzial entfalten wird. Die Erfolgsaussichten hängen wesentlich davon ab, ob Energieversorgungsunternehmen das Potenzial erkennen und versuchen, Blockchain-Anwendungen frühzeitig in bestehende Prozesse der Energiewirtschaft zu integrieren.

Literatur

[1] Adam Back et al. *Enabling Blockchain Innovations with Pegged Sidechains*. Blockstream, 2014.

[2] Alexander Bogensperger et al. *Die Blockchain-Technologie - Chance zur Transformation der Energieversorgung?* Forschungsstelle für Energiewirtschaft e.V. (FfE), 2018.

[3] Baraq Ghaleb et al. *Addressing the DAO Insider Attack in RPL's Internet of Things Networks*. IEEE Communications Letters, Ausgabe 23, 2019.

[4] Chao Liu et al. *Finding Reentrancy Bugs in Smart Contracts*. 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), 2018.

[5] Claudia Pop et al. *Blockchain and Demand Response: Zero-Knowledge Proofs for Energy Transactions Privacy*. Computer Science Department, Technical University of Cluj-Napoca, 2020.

[6] Florian Glatz et al. *Blockchain - Chancen und Herausforderungen einer neuen digitalen Infrastruktur für Deutschland*. Blockchain Bundesverband e.V., 2017.

[7] Frieder Schnabel et al. *Smarte umweltrelevante Infrastrukturen: Anwendungsfelder, Bedarfe, Praxiserfahrung aus kommunaler Sicht*. Umweltbundesamt, 2019.

[8] Gunter Arnold et al. *Intelligente Netzanbindung von Elektrofahrzeugen zur Erbringung von Systemdienstleistungen*. INEES Abschlussbericht, 2016.

[9] Holger Berndt et al. *Netz- und Systemregeln der deutschen Übertragungsnetzbetreiber*. Verband der Netzbetreiber VDN e.V., 2007.

[10] Igor Zikratov et al. *Ensuring data integrity using blockchain technology*. 20th Conference of Open Innovations Association, 2017.

[11] Lawrence Orsini et al. *How the Brooklyn Microgrid and TransActive Grid are paving the way to next-gen energy markets*. The Energy Internet, 2019.

[12] Loi Luu et al. *Making Smart Contracts Smarter*. `https://eprint.iacr.org/2016/633.pdf`. [Letzter Aufruf: 20.09.2020]. 2016.

[13] Martin Wietschel et al. *Auswirkung der Elektromobilität auf die Haushaltsstrompreise in Deutschland*. Fraunhofer-Institut für System- und Innovationsforschung ISI, 2018.

[14] Maximilian Wohrer et al. *Smart Contracts: Security Patterns in the Ethereum Ecosystem and Solidity*. International Workshop on Blockchain Oriented Software Engineering, 2018.

[15] Mayank Raikwar et al. *Trends in Development of Databases and Blockchain*. 2020 Seventh International Conference on Software Defined Systems, 2020.

[16] Mingchao Yu et al. *Coded Merkle Tree: Solving Data Availability Attacks in Blockchains*. International Conference on Financial Cryptography und Data Security, 2020.

[17] Neville Grech et al. *MadMax: Surviving Out-of-Gas Conditions in Ethereum Smart Contracts*. Proceedings of the ACM on Programming Languages, 2018.

[18] Oesterle et al. *Memorandum on design-oriented information systems research*. European Journal of Information Systems, 2010.

[19] Özyılmaz et al. *Addressing Blockchain Scaling by Chain Splitting*. Department of Computer Engineering, Bogazici University, Istanbul, 2019.

[20] Qiheng Zhou et al. *Solutions to Scalability of Blockchain: A Survey*. IEEE Access, 2020.

[21] Quanqing Xu et al. *Building an Ethereum and IPFS-Based Decentralized Social Network System*. 24th International Conference on Parallel und Distributed Systems, 2018.

[22] Sandra Hartmann et al. *Automobile Wertschöpfung*. Universität des Saarlandes, 2019.

[23] Sean Braithwaite et al. *Formal Specification and Model Checking of the Tendermint Blockchain Synchronization Protocol*. Interchain Foundation, 2020.

[24] Sill et al. *Blockchain Standards for Compliance and Trust*. IEEE Cloud Computing, 2017.

[25] Soo Young Park et al. *Decentralized Directed acyclic graph based DLT Network*. Proceedings of the International Conference on Omni-Layer Intelligent Systems, 2019.

[26] Stefano Bistarelli et al. *Analysis of Ethereum Smart Contracts and Opcodes*. An Organized Repository of Ethereum Smart Contracts' Source Codes und Metrics, 2020.

[27] Susen Döbelt et al. *Access control for a Smart Grid SOA*. 2012 International Conference for Internet Technology und Secured Transactions, 2013.

[28] Thomas Bründlinger et al. *dena-Leitstudie Integrierte Energiewende: Impulse für die Gestaltung des Energiesystems bis 2050*. Deutsche Energie-Agentur GmbH, 2009.

[29] Wickert et al. *Wissenschaftliche Unterstützung bei der Erstellung von Fahrzeugbezogenen Analysen zur Netzintegration von Elektrofahrzeugen unter Nutzung Erneuerbarer Energien*. Das Fraunhofer-Institut für Windenergie und Energiesystemtechnik, 2017.

[30] Xu et al. *A Taxonomy of Blockchain-Based Systems for Architecture Design*. IEEE International Conference on Software Architecture, 2017.

[31] The open Application network. *Working together towards a common standard*. `https://aion.theoan.com/blog/blockchain-interoperability-alliance/`. [Letzter Aufruf: 20.02.2021]. 2021.

[32] L. M. Bach, B. Mihalejevic und M. Zagar. *Comparative Analysis of Blockchain Consensus Algorithms*. Rochester Institute of Technology, Croatia, 2018.

[33] Adam Back. *Hashcash - A Denial of Service Counter-Measure*. Cypherspace, 2002.

[34] Ayrat Badykov. *Message calls in Ethereum*. `https://hackernoon.com/getting-deep-into-evm-how-ethereum-works-backstage-ac7efa1f0015`. [Letzter Aufruf: 25.01.2021]. 2018.

[35] blocknet. *What is blocknet? - Technical Overview*. `https://docs.blocknet.co/`. [Letzter Aufruf: 20.01.2021]. 2021.

[36] Richard Gendal Brown. *Corda: An Introduction*. Corda, 2016.

[37] Bundesnetzagentur. *Stromerzeugung und Stromhandel 2020*. `https://www.smard.de/page/home/topic-article/444/202398`. [Letzter Aufruf: 05.03.2021]. 2021.

[38] Vitalik Buterin. *A Next Generation Smart Contract & Decentralized Application Platform*. Ethereum Foundation, 2014.

[39] Vitalik Buterin und Virgil Griffith. *Casper the Friendly Finality Gadget*. `https://arxiv.org/abs/1710.09437`. [Letzter Aufruf: 20.02.2021]. 2021.

[40] Christian Cachin. *Architecture of the Hyperledger Blockchain Fabric*. IBM Research, 2016.

[41] Chainlink. *Chainlink VRF: On-chain Verifiable Randomness*. `https://blog.chain.link/verifiable-random-functions-vrf-random-number-generation-rng-feature/`. [Letzter Aufruf: 24.11.2020]. 2020.

[42] Huashan Chen u. a. *A Survey on Ethereum Systems Security: Vulnerabilities, Attacks and Defenses*. `https://arxiv.org/pdf/1908.04507.pdf`. [Letzter Aufruf: 26.11.2020]. 2020.

[43] Pragmatic Coders. *How (not) to screw up Solidity smart contract*. `https://pragmaticcoders.com/blog/solidity-smart-contract/`. [Letzter Aufruf: 26.11.2020]. 2020.

[44] Andreas Corusa, Johannes Predel und Nikolas Schöne. *Eine Marktübersicht der Blockchain in der Energiewirtschaft*. Technische Universität Berlin Institut für Energietechnik, 2020.

[45] Chris Coverdale. *Solidity: Transaction-Ordering Attacks*. `https://medium.com/coinmonks/solidity-transaction-ordering-attacks-1193a014884e`. [Letzter Aufruf: 23.11.2020]. 2018.

[46] Eric Demuth, Paul Klanschek und Christian Trummer. *Was ist eine Hash-Funktion in einer Blockchain-Transaktion?* `https://www.bitpanda.com/academy/de/lektionen/was-ist-eine-hash-funktion-in-einer-blockchain-transaktion/`. [Letzter Aufruf: 18.01.2021]. 2020.

[47] digiconomist. *Bitcoin Energy Consumption Index)*. `https://digiconomist.net/bitcoin-energy-consumption/`. [Letzter Aufruf: 27.10.2020]. 2020.

[48] ConsenSys Diligence. *Ethereum Smart Contract Best Practices: Known Attacks*. `https://consensys.github.io/smart-contract-best-practices/known_attacks/`. [Letzter Aufruf: 12.09.2020]. 2018.

[49] DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik in DIN und VDE. *Der Technische Leitfaden - Ladeinfrastruktur Elektromobilität*. BDEW, 2020.

[50] Hyperledger Fabric Documentation. *Using a Hardware Security Module (HSM)*. `https://hyperledger-fabric.readthedocs.io/en/release-2.2/hsm.html#`. [Letzter Aufruf: 28.12.2020].

[51] Jordan Earls. *The Faults and Shortcomings of the EVM*. `https://medium.com/@earlz/the-faults-and-shortcomings-of-the-evm-bde4d09b8b6a`. [Letzter Aufruf: 18.11.2020]. 2017.

[52] Arbeitsgemeinschaft Mess- und Eichwesen. *Eichrechtliche Grundlagen im Bereich der Elektromobilität*. Deutsche Akademie für Metrologie, 2016.

[53] ethereum.org. *Ethereum - a global, open-source platform for decentralized applications*. `https://ethereum.org/en/`. [Letzter Aufruf: 18.11.2020]. 2020.

[54] ethereum.org. *ETHEREUM VIRTUAL MACHINE (EVM)*. `https://ethereum.org/en/developers/docs/evm/`. [Letzter Aufruf: 29.01.2021]. 2020.

[55] ethereum.org. *Upgrading Ethereum to radical new heights*. `https://ethereum.org/en/eth2/beacon-chain/`. [Letzter Aufruf: 18.01.2021]. 2021.

[56] Energy Web Foundation. *Accelerating the Energy Transition with an Open-Source, Decentralized Blockchain Platform)*. https://energyweb.org/wp-content/uploads/2019/05/EWF-Paper-TheEnergyWebChain-v2-201907-FINAL.pdf. [Letzter Aufruf: 02.10.2020]. 2019.

[57] Ethereum Foundation. *Contracts*. https://docs.soliditylang.org/en/v0.5.3/contracts.html. [Letzter Aufruf: 26.01.2021]. 2020.

[58] Ethereum Foundation. *Conventions in Solidity*. https://solidity.readthedocs.io/. [Letzter Aufruf: 13.01.2021]. 2020.

[59] Ethereum Foundation. *Ethereum WebAssembly (ewasm)*. https://ewasm.readthedocs.io/en/mkdocs/. [Letzter Aufruf: 13.11.2020]. 2020.

[60] Ethereum Foundation. *Security Considerations*. https://docs.soliditylang.org/en/develop/security-considerations.html. [Letzter Aufruf: 27.01.2021]. 2021.

[61] Ethereum Foundation. *Solidity*. https://solidity.readthedocs.io/. [Letzter Aufruf: 18.11.2020]. 2020.

[62] Ethereum Foundation. *Solidity*. https://docs.soliditylang.org/en/v0.4.24/. [Letzter Aufruf: 24.11.2020]. 2020.

[63] Ethereum Foundation. *Solidity Value Types*. https://docs.soliditylang.org/en/v0.4.21/types.html. [Letzter Aufruf: 10.12.2020]. 2020.

[64] Ethereum Foundation. *Units and Globally Available Variables*. https://docs.soliditylang.org/en/v0.4.21/units-and-global-variables.html. [Letzter Aufruf: 13.01.2021]. 2020.

[65] *Gesetz zur Digitalisierung der Energiewende*. Bundesministerium für Wirtschaft und Energie, 2016.

[66] Amprion GmbH. *Höchstspannung garantiert*. https://www.amprion.net/Netzjournal/Beitr%C3%A4ge-2019/H%C3%B6chstspannung-garantiert.html. [Letzter Aufruf: 27.11.2020].

[67] Sonnen GmbH. *Eine Gemeinschaft von Unabhängigen*. https://sonnen.de/sonnencommunity/. [Letzter Aufruf: 05.02.2021]. 2021.

[68] WA Notstromtechnik GmbH. *Stromerzeuger Lexikon*. https://www.stromerzeuger-lexikon.de/grundlast/. [Letzter Aufruf: 27.11.2020].

[69] Till Gnann. *Market Diffusion of Plug-in Electric Vehicles and their Charging Infrastructure*. Fraunhofer Institute for Systems und Innovation Research ISI, 2015.

[70] Ilya Grishchenko, Matteo Maffei und Clara Schneidewind. *A Semantic Framework for the Security Analysis of Ethereum Smart Contracts*. International Conference on Principles of Security und Trust, 2018.

[71] NCC Group. *Decentralized Application Security Project*. `http://dasp.co/`. [Letzter Aufruf: 21.09.2020]. 2018.

[72] Stromnetz Hamburg. *So kommt der Strom zu Ihnen*. `https://www.stromnetz-hamburg.de/ueber-uns/aufgaben/verteilungsnetzbetreiber`. [Letzter Aufruf: 06.10.2020]. 2020.

[73] Sandra Hook. *Einführung in die Regenerative Energiewirtschaft*. Springer Verlag, 2019.

[74] Lorenz Jarass, Gustav Obermair und Wilfried Voigt. *Windenergie: Zuverlässige Integration in die Energieversorgung*. Springer Science Business Media, 2009.

[75] Anna Kobylinska. *Mit Sharding zu mehr Performance in der Blockchain*. `https://www.blockchain-insider.de/mit-sharding-zu-mehr-performance-in-der-blockchain-a-855675/`. [Letzter Aufruf: 07.03.2021].

[76] Juri Mattila. *The Blockchain Phenomenon - The Disruptive Potential of Distributed Consensus Architectures*. Research Institute of the Finnish Economy (ETLA), 2016.

[77] Brooklyn Microgrid. *Brooklyn Microgrid Overview*. `https://www.brooklyn.energy/about`. [Letzter Aufruf: 05.02.2021]. 2021.

[78] Jelena Mišić, Vojislav Mišić und Xiaolin Chang. *On the Benefits of Compact Blocks in Bitcoin*. IEEE International Conference on Communications, 2020.

[79] Mitchell, Bradley R. Agle und Donna J. Wood. *Toward a theory of stakeholder identification and salience: Defining the principle of who and what really counts*. Academy of Management, 1997.

[80] Michiel Mulders. *Smart Contract Safety: Best Practices Design Patterns*. `https://www.sitepoint.com/smart-contract-safety-best-practices-design-patterns/`. [Letzter Aufruf: 28.01.2021]. 2018.

[81] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Bitcoin Foundation, 2008.

[82] Satoshi Nakamoto. *repository of Satoshi Nakamoto's original bitcoin sourcecode)*. `https://github.com/trottier/original-bitcoin`. [Letzter Aufruf: 01.10.2020]. 2009.

[83] Bundesministerium für Umwelt Naturschutz und nukleare Sicherheit. *Klimaschutz in Zahlen*. 2020.

[84] Till Neudecker und Hannes Hartenstein. *Short Paper: An Empirical Analysis of Blockchain Forks in Bitcoin.* International Conference on Financial Cryptography und Data Security, 2019.

[85] Jeff Nijsse und Alan Litchfield. *A Taxonomy of Blockchain Consensus Methods.* Service und Cloud Computing Research Lab, 2020.

[86] OpenZeppelin. *ReentrancyGuard - protects you from reentrant calls.* `https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/ReentrancyGuard.sol`. [Letzter Aufruf: 12.01.2021].

[87] Solidity Patterns. *A compilation of patterns and best practices for the smart contract programming language Solidity.* `https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html`. [Letzter Aufruf: 23.11.2020]. 2018.

[88] Anja Peters und Elisabeth Dütschke. *Zur Nutzerakzeptanz von Elektromobilität.* Fraunhofer Systemforschung Elektromobilität FSEM, 2010.

[89] Peters und Mohr. *Digitalisierung im Energiemarkt: Neue Chancen, neue Herausforderungen.* Energiewirtschaftliche Tagesfragen, 2016.

[90] Ante Plazibat. *Blockchain, mehr als nur ein Hype? – Eine Einführung in die Blockchain.* `https://ccecosystems.news/blockchain-mehr-als-nur-ein-hype-eine-einfuehrung-in-die-blockchain/`. [Letzter Aufruf: 18.01.2021]. 2016.

[91] Joseph Poon und Thaddeus Dryja. *Scalable Off-Chain Instant Payments.* The Bitcoin Lightning Network, 2016.

[92] Ethereum Smart Contract Best Practices. *Secure Development Recommendations.* `https://consensys.github.io/smart-contract-best-practices/recommendations/#timestamp-dependence`. [Letzter Aufruf: 24.11.2020]. 2020.

[93] Matevž Pustišeka und Andrej Kosa. *Approaches to Front-End IoT Application Development for the Ethereum Blockchain.* 2017 International Conference on Identification, Information und the Internet of Things, 2017.

[94] randao. *Random number in programming is very important!* `https://github.com/randao/randao`. [Letzter Aufruf: 13.01.2021].

[95] Europäisches Parlament und Rat der Europäischen Union. *Datenschutz-Grundverordnung.* `https://dsgvo-gesetz.de/`. [Letzter Aufruf: 06.03.2021]. 2018.

[96] Ines Roth. *Digitalisierung in der Energiewirtschaft.* Hans Böckler Stiftung, 2018.

[97] Ingo Rübe. *Zukunftsfähige Rahmenbedingungen für die Distributed-Ledger-Technologie im Finanzmarkt*. https://www.bundestag.de/resource/blob/627992/62298d2fd5b3497f5fca4d248 03-BOTLabs-GmbH-data.pdf. [Letzter Aufruf: 10.02.2021]. 2021.

[98] Vaibhav Saini. *Getting Deep Into EVM: How Ethereum Works Backstage*. https://hackernoon.com/getting-deep-into-evm-how-ethereum-works-backstage-ac7efa1f0015. [Letzter Aufruf: 25.01.2021]. 2018.

[99] Jörg Schmidt und Michael Wagner. *Reise in eine klimaschonende Zukunft – Energiewende im Wärmemarkt*. Springer Fachmedien Wiesbaden GmbH, 2018.

[100] Alexander Schuller. *Marktintegration der Elektromobilität: Ein agentenbasierter Ansatz für das Smart Grid*. Karlsruher Institut für Technologie, 2010.

[101] Adolf J. Schwab. *Elektroenergiesysteme - Erzeugung, Transport, Übertragung und Verteilung elektrischer Energie*. Institut Elektroenergiesysteme und Hochspannungstechnik Universität Karlsruhe, 2017.

[102] SEDC. *Explicit and Implicit Demand-Side Flexibility*. Smart Energy Demand Coalition, 2016.

[103] Khrystyna Shakhmatova. *SMART Capital Region 2.0*. http://www.smartcapitalregion.de/download/Projektskizze_SCR%202.0.pdf. [Letzter Aufruf: 05.03.2021]. 2019.

[104] Solidity. *Solidity Releases*. https://github.com/ethereum/solidity/releases. [Letzter Aufruf: 27.01.2021]. 2021.

[105] Joachim Specovius. *Grundkurs Leistungselektronik, 10. Auflage*. Springer Verlag, 2020.

[106] Nick Szabo. *Smart Contracts: Building Blocks for Digital Markets*. 1996.

[107] Paolo Tasca und Claudio J. Tessone. *Taxonomy of Blockchain Technologies. Principles of Identification and Classification*. Centre for Blockchain Technologies University College London, 2017.

[108] VDE. *Die Deutsche Normungs-Roadmap*. VDE Verband der Elektrotechnik Elektronik Informationstechnik e.V., 2017.

[109] Arun Kumar Verma, Bhim Singh und D.T. Shahani. *Grid to vehicle and vehicle to grid energy transfer using single-phase bidirectional AC-DC converter and bidirectional DC-DC converter*. Proceedings of the international conference on energy, automation und signal, 2011.

[110] Franz Volland. *solidity-patterns*. https://fravoll.github.io/solidity-patterns/pull_over_push.html. [Letzter Aufruf: 04.01.2021]. 2021.

[111] Christof Weinhardt. *Landau Microgrid Project (LAMP)*. `https://im.iism.kit.edu/news_2098.php`. [Letzter Aufruf: 28.01.2021]. 2017.

[112] Kevin Werbach. *The Blockchain and the New Architecture of Trust*. massachusetts institute of technology, 2018.

[113] Jakub Zakrzewski. *Towards Verification of Ethereum Smart Contracts: A Formalization of Core of Solidity*. Working Conference on Verified Software: Theories, Tools, und Experiments, 2018.

[114] Hendrik Zimmermann und Janna Hoppe. *Chancen und Risiken der Blockchain für die Energiewende*. Germanwatch e.V., 2018.

[115] Guy Zyskind, Oz Nathan und Alex Pentland. *Decentralizing Privacy: Using Blockchain to Protect Personal Data*. IEEE CS Security und Privacy Workshops, 2015.

Abbildungsverzeichnis