Fakultät für Ingenieurwissenschaften

Master-Thesis

Untersuchungen zum Einsatz von Blockchain Technologien für das Internet der Dinge

Abschlussarbeit zur Erlangung des Grades eines

Master of Engineering

der Hochschule Wismar

eingereicht von:	Detlev Noll
	geboren am 06. Juli 1966 in Dierdorf
	Studiengang IT-Sicherheit und Forensik
Matrikelnummer:	197766
Erstgutachter:	Prof. Dr.-Ing. A. Ahrens
Zweitgutachter:	Prof. Dr.-Ing. A. Raab-Düsterhöft

Wismar, den 28. Juni 2018

Vorwort

An dieser Stelle möchte ich Herr Prof. Dr. Ahrens für die Betreuung der Arbeit danken. Er hat mich stets konstruktiv und motivierend unterstützt und mit seinen wertvollen Anregungen deutlich zum Gelingen dieser Arbeit beigetragen.
Frau Prof. Dr. Raab-Düsterhöft danke ich für die freundliche Übernahme des Korreferats.

Inhaltsverzeichnis

Bilderverzeichnis **VI**

Tabellenverzeichnis **VII**

Quellcodeverzeichnis **VII**

Abkürzungsverzeichnis **IX**

1. Einleitung, Motivation und Zielsetzung **1**

2. Das Internet der Dinge **3**

3. Blockchain Technologien **6**

3.1. Unveränderbarkeit von Daten in der Blockchain 7

3.1.1. Merkle-Trees 7

3.1.2. Hash-Kette 8

3.1.3. Die Blockchain-Datenstruktur 9

3.2. Identifikation, Authentifikation und Authorisierung in der Blockchain 9

3.3. Vertrauen in der Blockchain 10

3.3.1. Proof of Work 11

3.3.2. Peer-to-Peer Blockchain-Netzwerk 12

3.3.3. Consensus Algorithmus in der Blockchain 13

3.4. Blockchain Varianten 15

3.4.1. Alternative Consensus Verfahren 16

3.4.2. Blockchains mit eingeschränktem Nutzerkreis 17

3.5. Smart Contracts 19

3.5.1. Smart Contracts in der Ethereum Blockchain 19

3.5.2. Smart Contracts in permissioned Blockchains 22

3.6. Zusammenfassung 24

4. Einsatzmöglichkeiten von Blockchains für das Internet der Dinge **25**

4.1. Grundsätzliche Einsatzmöglichkeiten 25

4.2. Konkrete Anwendungsszenarien 27

5. Simulation ausgewählter Anwendungsszenarien **31**
5.1. Entwicklungs- und Simulationsumgebung 31
5.1.1. Private Simulations-Blockchain 31
5.1.2. Smart Contract Entwicklung 32
5.1.3. Administration der Blockchain Accounts und Verträge 32
5.1.4. Simulation des IoT Endgerätes 32
5.1.5. Anbindung von simulierten IoT Endgeräten 33
5.1.6. Web3 ABI 35
5.1.7. IPFS 35
5.2. Szenario 1: Firmware Updates für IoT Geräten 35
5.2.1. Szenariosetup 35
5.2.2. Smart Contract 36
5.2.3. IoT Simulation 37
5.2.4. Simulationslauf 40
5.2.5. Feststellungen 41
5.3. Szenario 2: Aggregation von Sensordaten 42
5.3.1. Szenariosetup 42
5.3.2. Smart Contract 43
5.3.3. IoT Simulation 45
5.3.4. Simulationslauf 46
5.3.5. Auswertung der Simulation 46
5.3.6. Feststellungen 49

6. Ergebnisse und Diskussion **51**
6.1. Technologisch bedingte Einschränkungen 52
6.1.1. Vertraulichkeit 52
6.1.2. Skalierbarkeit 53
6.1.3. Validierungszeit 53
6.1.4. De-Dezentralisierung 54
6.2. IT Sicherheitsrisiken der Blockchain 54
6.2.1. Schlüsselmanagement 54
6.2.2. Korrektheit von Smart Contracts 55
6.2.3. Fehler in der Implementierung des Blockchain Protokolls . . . 56
6.2.4. Bekannte Angriffsvektoren 56
6.3. Übergreifende Aspekte 57

7. Zusammenfassung **60**

Literatur **61**

Online-Quellen **69**

A. Listings zur Simulationsumgebung **71**

B. Listings zur Simulation des Szenarios 1 **72**

C. Listings zur Simulation des Szenarios 2 **74**

D. Verzeichnis der digitalen Anlagen **88**

Bilderverzeichnis

2.1. IoT Technologiestack und beteiligte Akteure 4

3.1. Merkle Hash Tree 8
3.2. Hash-Kette 9
3.3. Blockchain-Datenstruktur 10

5.1. Anbindungsoptionen von IoT Geräten an die Blockchain 34
5.2. Im Mist Browser angezeigte Transaktionsinformationen zur Anlage des Vertrages und zum Update der Versionsinformationen. 37
5.3. Exemplarischer Auszug des Abgleichs von Sensorlog und Eventlog. . 48

6.1. IoT Technologiestack mit Blockchain Layer und beteiligte Akteure . . 58

Tabellenverzeichnis

3.1. Transaktionen Smart Contract Beispiel 22

5.1. Ausschnitt des Sensorlogs in tabellarischer Form 47
5.2. Beispielauszüge des Eventlogs 48

D.1. Digitale Anlagen 88

Quellcodeverzeichnis

3.1. Pseudocode eines einfachen Smart Contracts 21

5.1. Smart Contract `LatestFirmware.sol` 36
5.2. Simulierte Firmware-Datei `firmware.txt` 37
5.3. Firmware Status auf dem IoT Gerät in `current_fw.json` 38
5.4. Python Script `fw.py` zur Simulation des IoT Endgerätes 39

A.1. `genesis.json` Datei zur Initialisierung der privaten BC 71

B.1. JSON ABI des Smart Contracts `LatestFirmware.sol`. 72

C.1. Smart Contract `Sensorwarner7.sol` 74
C.2. JSON ABI zum Smart Contract `Sensorwarner7.sol` 77
C.3. Python Skript `gen_accounts.py` 79
C.4. Python Skript `initial_ether.py` 79
C.5. Python Skript `register_sensors.py` 80
C.6. Python Skript `sim_sensor.py` 80
C.7. Python Skript `parallel.py` 82
C.8. Python Skript `eventlog.py` 82
C.9. Python Skript `get_sensor_status.py` 83
C.10. Datei `sensor_accounts.json` 84
C.11. Sensorstatus nach Beendigung der Simulation (Output von `get_sensor_status.py`) 84

Abkürzungsverzeichnis

ABI	Application Binary Interface
API	Application Programming Interface
BC	Blockchain
CPS	Cyber-Physische Systeme
DAPP	Distributed Application
DLT	Distributed Ledger Technologie
DoS	Denial of Service
EVM	Ethereum Virtual Machine
FN	Full Node
IoT	Internet der Dinge (englisch: Internet of Things)
IP	Internet Protokoll
IPFS	InterPlanetary File System
LN	Light Node
RPC	Remote Procedure Call
TEE	Trusted Execution Environment
TRU	tracable ressource unit
USN	Universal Sharing Network
VM	virtuelle Maschine
WWW	World Wide Web

1. Einleitung, Motivation und Zielsetzung

Die stetig wachsende Digitalisierung ermöglicht immer neue Anwendungen und Geschäftsmodelle auf Basis vernetzter Prozesse, Produkte, Geräte, Maschinen und den Informationen, die sie sammeln, auswerten, nutzen und teilen. Diese Entwicklung schlägt sich insbesondere in den Konzepten vom Internet der Dinge (englisch: Internet of Things, IoT) und der Zusammenführung von physischen Systemen und virtuellen Objekten zu Cyber-Physischen Systemen (CPS) als Grundlage der Industrie 4.0 nieder. Neben den sich hierdurch ergebenden Chancen und Möglichkeiten steigen auch die Risiken, indem sich durch das Zusammenwachsen der physischen und der virtuellen Welten neue Angriffsmöglichkeiten ergeben. Im Hinblick auf die Informationssicherheit stellen sich darüber hinaus Fragen, wie in zunehmend dezentral agierenden Systemen die Vertraulichkeit, die Nichtabstreitbarkeit oder die Integrität von Informationen gewährleistet werden können.

Ein möglicher Ansatzpunkt in diesem Kontext sind Blockchain-Technologien [Eck17]. Zunächst populär wurde dieses Konzept durch die Einführung der Bitcoins [Nak08], bei denen die Blockchain (BC) als dezentrale Datenbank genutzt wird und als Verzeichnis aller Bitcoin-Transaktionen[1] dient. Der Begriff Blockchain leitet sich hierbei maßgeblich aus der Art der technischen Realisierung ab. Mehrere Transaktionen werden zu einem Block zusammengefasst und kryptographisch abgesichert. In einen Block wird zusätzlich jeweils Wissen über den vorhergehenden Block eingebunden. Alle Blöcke und somit alle Transaktionen sind damit unveränderbar miteinander verkettet.

Aus Anwendungssicht wird häufig auch der Begriff Distributed Ledger Technologie (DLT) verwendet. Hierin drückt sich aus, dass es bei Einsatz dieser Technologie nicht zwangsläufig einer zentralen Instanz bedarf, der vertraut werden muss, um die Korrektheit der Informationen zu bestätigen. Insbesondere dieser Umstand lässt die Blockchain Technologie grundsätzlich als attraktiv für die Nutzung im dezentralen Umfeld

[1] Eine Transaktion im technischen Sinne ist eine Folge von Operationen, die nur vollständig oder gar nicht ausgeführt werden. Eine Transaktion im wirtschaftlichen Sinne ist die gegenseitige Übertragung von virtuellen oder physischen Werten. Eine technische Transaktion kann eine wirtschaftliche Transaktion implizieren. Beispielsweise überträgt eine Transaktion in der Bitcoin Blockchain geldwerte Tokens (die Bitcoins) von einem Account auf einen anderen Account.

von IoT und der Industrie 4.0 erscheinen.
Ziel der vorliegenden Arbeit ist es, einerseits das Potential zu diskutieren, dass sich aus der Kombination der Technologiekonzepte IoT und Blockchain für Anwendungen ergibt, andererseits aber auch damit verbundene mögliche Risiken und Einschränkungen zu identifizieren. Kapitel 2 betrachtet hierzu zunächst die grundlegenden IoT Konzepte und die damit einhergehenden besonderen Probleme im Hinblick auf die Informationssicherheit. Kapitel 3 stellt dann die Grundlagen der Blockchain Technologien dar und motiviert deren Nutzung im Umfeld von IoT. Kapitel 4 diskutiert verschiedene Anwendungsmöglichkeiten von Blockchain Technologien für das IoT. Zwei dieser Anwendungsfälle werden in Kapitel 5 in Form einer Simulation nachgestellt, um einerseits die grundsätzliche Machbarkeit zu prüfen, andererseits aber auch Punkte zu identifizieren, die zum aktuellen Zeitpunkt einer praktischen Anwendung in einem produktiven Umfeld möglicherweise noch entgegenstehen. Kapitel 6 diskutiert und bewertet die Ergebnisse, die Arbeit schließt mit einer kurzen Zusammenfassung in Kapitel 7.

2. Das Internet der Dinge

Der Definition der International Telecommunication Unit aus 2012 folgend ist das Internet der Dinge „eine globale Infrastruktur für die Informationsgesellschaft, die fortschrittliche Dienste ermöglicht, indem physische und virtuelle Dinge auf der Grundlage bestehender und sich entwickelnder interoperabler Informations- und Kommunikationstechnologien miteinander verknüpft werden“ [ITU12]. Ein Ding ist hierbei „ein Objekt der physischen Welt (physisches Ding) oder der Informationswelt (virtuelles Ding), das in Kommunikationsnetzwerken identifiziert und integriert werden kann“.
Damit erweitert das IoT unsere Kommunikationsmöglichkeiten um eine zusätzliche Dimension. Neben der Möglichkeit, „jederzeit“ und „überall“ miteinander zu kommunizieren, können auf Basis dieser Definition nun auch Informationen „zwischen jedem und allem“ ausgetauscht werden.
Dies eröffnet zahlreiche neue Anwendungsoptionen. Nutzungsbeispiele im Endanwenderumfeld sind Smart Home Applikationen, Fitness-Tracker oder Wearables. Im industriellen Umfeld schlagen sich IoT Konzepte häufig in Anwendungen nieder, die die Attribute „connected“ oder „smart“ tragen: Connected Industry, Connected Car, Connected Building, Connected Health, Smart City oder Smart Energy sind nur eine Auswahl.
Mit den Vorteilen, die diese Lösungen fraglos für den Einzelnen, Unternehmen, Organisationen oder die Gemeinschaft haben, sind auch gestiegene Sicherheitsrisiken verbunden. Durch die Vielzahl der vernetzten Objekte steigt die Anzahl der möglichen Angriffspunkte sowohl für virtuellen und auch physischen Zugriff deutlich. Jedes einzelne Ding selbst kann Schwachstellen in sich führen und der IoT Technologie-Stack als Kombination aus Hardware, Software und Kommunikationsmitteln ist hochgradig heterogen (vergleiche Bild 2.1). Auch sind in der Regel eine Vielzahl von Akteuren an der Implementierung und dem Betrieb eines IoT-Dienstes beteiligt.
Die potentiellen Auswirkungen von Angriffen steigen, da beispielsweise als Ding eingebundene Aktoren bei missbräuchlicher Anwendung direkte physische Schäden verursachen können. Ferner befinden sich IoT Komponenten häufig auch in nicht zugangsgesicherten Bereichen und erlauben potentiellen Angreifern auch den physischen Zugriff.
Als besondere Herausforderungen im Hinblick auf die IT-Sicherheit stellen sich daher

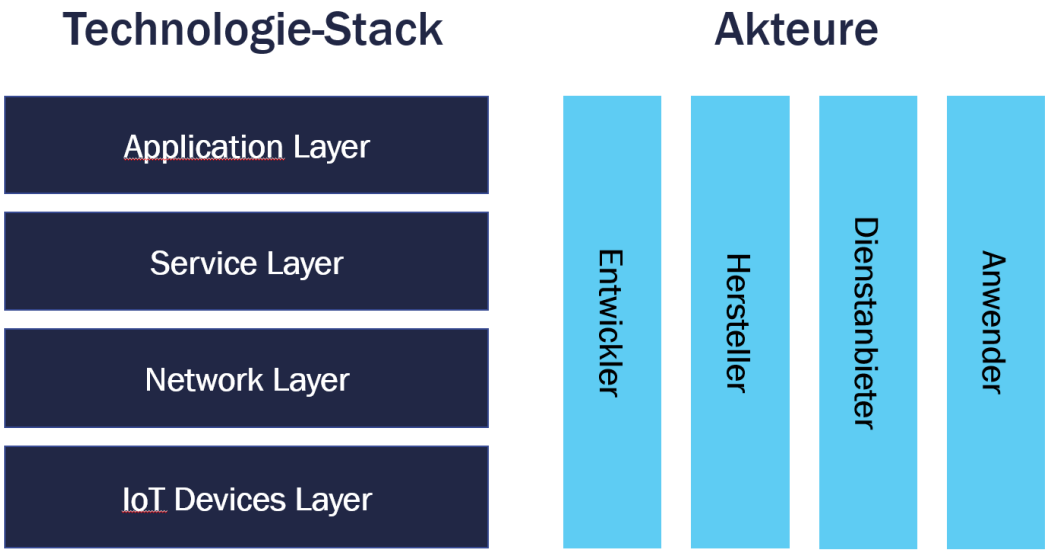


Bild 2.1.: IoT Technologiestack und beteiligte Akteure

- die Sicherstellung der Kontrolle über die vernetzten Dinge,
- die Wahrung der Hoheit über die Daten und
- der Schutz von Schlüsselmaterial und Zugangsdaten.

So listet z.B. Maple in seinem Übersichtsartikel [Map17] die folgenden sicherheitsrelevanten Problemfelder:

- Physische Einschränkungen von Geräten und Kommunikation,
- Heterogenität, Skalierung und ad-hoc Natur,
- Authentisierung und Identitätsmanagement,
- Authorisierung und Zugriffskontrolle sowie
- Implementierung, Updates, Verantwortung und Rechenschaftspflicht.

Den klassischen Sicherheitsrisiken und Angriffsvektoren kann durch die konsequente Nutzung von Security Frameworks begegnet werden, einen Überblick gibt z.B. [Irs16]. Daneben muss aber auch Aspekten des Datenschutzes im Kontext von IoT besondere Beachtung geschenkt werden. Es werden eine Vielzahl von z.T. sehr sensiblen Daten generiert, und während diese Daten im klassischen World Wide Web (WWW) im Wesentlichen als personenbezogene Daten Einzelnen gehören, werden in IoT Systemen auch Daten generiert, die z.B. der Gemeinschaft, Gruppen oder Organisationen zuzurechnen sind.

Vertrauen in die Dienste, die diese Daten nutzen und verarbeiten, setzt Kontrolle und Transparenz darüber voraus, wann welche Daten mit wem zu welchem Zweck geteilt wurden. Und Vertrauen erfordert die Integrität der Daten selbst, aber auch die Integrität der Protokolle über deren Nutzung in Bezug auf Vollständigkeit, Konsistenz und Korrektheit. Dies sind – wie der folgende Abschnitt erläutert – inherente Eigenschaften, die von Blockchains bereitgestellt werden.

3. Blockchain Technologien

Die Verwendung des Begriffes Blockchain ist nicht eindeutig. Eine Blockchain-Systemarchitektur nutzt einen Blockchain-Algorithmus, der auf einer Blockchain-Datenstruktur operiert. Aus dem Zusammenhang ist in der Regel ersichtlich, auf welchen Aspekt jeweils Bezug genommen wird. Mittels einer Blockchain-Architektur kann eine verteilte Datenbankapplikation realisiert werden mit den folgenden Eigenschaften [Dre17]:

1. Einmal abgelegte Daten sind unveränderlich.
2. Daten können nur hinzugefügt, nicht gelöscht werden.
3. Die Daten sind geordnet und besitzen einen Zeitstempel.
4. Die Daten sind offen zugänglich.
5. Transaktionen in der Datenbank sind kryptographisch abgesichert (Identifikation, Authentifikation und Authorisierung).
6. Die Daten sind (nicht zu jedem Zeitpunkt, aber letztlich) konsistent.
7. Auf die Integrität der Daten kann systembedingt vertraut werden. Das Vertrauen in einen Mittelsmann ist nicht erforderlich.

Mit einem solchen System ist es möglich,

- die Existenz oder Nichtexistenz eines bestimmten Dateneintrages,
- den Zeitpunkt und die Reihenfolge von Einträgen,
- den Initiator einer Transaktion und
- den Eigentümer eines Dateneintrages

zweifelsfrei zu beweisen, ohne einem Mittelsmann vertrauen zu müssen. Dokumentieren die Dateneinträge Eigentum an Sachen oder sonstigen Werten und sind die Transaktionen auf der Datenstruktur Eigentumsübertragungen, dann genügt eine solche Anwendung den strengen Anforderungen einer Buchführung. Daher werden Blockchains auch als Distributed Ledger Technologien (DLT) bezeichnet.

In seinem Beitrag [Nak08] aus dem Jahr 2008 hat Satoschi Nakamoto (ein Pseudonym, die wahre Identität des Autors ist bis heute öffentlich nicht bekannt) die Architektur eines solchen Systems beschrieben für den Handel mit der digitalen Währung Bitcoin. Er erzielt die geforderten Eigenschaften an das System dadurch, dass er verschiedene, teils schon seit vielen Jahren bekannte, Verfahren und Konzepte in geschickter Weise kombiniert. Im folgenden werden zunächst die von ihm genutzten technologischen Basiskonzepte und dann ihr Zusammenwirken erläutert. Die Bitcoin Blockchain Architektur bildet die Grundstruktur, auf der sich verschiedene Varianten mit unterschiedlichen Einsatzschwerpunkten entwickelt haben. Diese Varianten und Erweiterungen werden in Folge vorgestellt.

3.1. Unveränderbarkeit von Daten in der Blockchain

3.1.1. Merkle-Trees

Bereits 1980 stellte Merkle dieses in 1982 patentierte Verfahren zur Signatur einer Vielzahl von Datenpaketen in Form eines Hash [1] Baumes vor [Mer80; Mer82].
Seien $2n$ einzelne Datenpakete D_i gegeben. Diese Datenpakete bilden die Blätter eines binären Baumes [2]. Für jedes Datenpaket wird ein Hash Wert $H_i = h(D_I)$ ermittelt. Alle inneren Knoten des Hash Baumes ergeben sich dann als Hash-Wert über die Hash-Werte der Nachfolger des Knotens. Verdeutlicht ist dies in Bild 3.1. So ergibt sich in dieser Anordnung $H_{12} = h(H_1, H_2)$ als Hash-Wert über die verketteten Einträge H_1 und H_2, deren Werte selbst die Hash-Werte der diesen Blättern anhängenden Datenpakete

[1] Ein Hash-Wert H ist der Ausgabewert einer determinitischen kryptographischen Funktion h, die eine beliebige Menge von Eingangsdaten D auf eine Menge fixer Länge (N Bit) abbildet:

$$H = h(D) \text{ mit } H < 2^N. \tag{3.1}$$

Für kryptographische Anwendungen muss eine solche Hash-Funktion folgende Eigenschaften aufweisen:

1. Die Funktion h ist eine Einwegfunktion. Es existiert keine einfach zu berechnende Umkehrfunktion und es ist im Allgemeinen nicht möglich, aus einem Hashwert H die Daten D zu ermitteln.
2. Eine kleine Änderung in den Daten D führt zu einer großen Änderung im Hash-Wert H.

Ein solcher Hashwert $H = h(D)$ kann als „Fingerabdruck“ der Daten interpretiert werden, über den die Daten verifiziert werden können. Für Details zu verschiedenen Hash-Funktionen und deren speziellen Eigenschaften wird auf [Jan16] verwiesen.

[2] Das Verfahren ist grundsätzlich auch anwendbar unter Nutzung von Bäumen, deren Knoten jeweils $N > 2$ Nachkommen haben. Die Beschränkung auf Binärbäume dient nur der Erläuterung des Grundprinzips.

sind. Die Wurzel dieses Hash Baumes wird als Merkle Root und der Eintrag dieses Wurzelknotens als Merkle-Root-Hash H_{MR} bezeichnet.

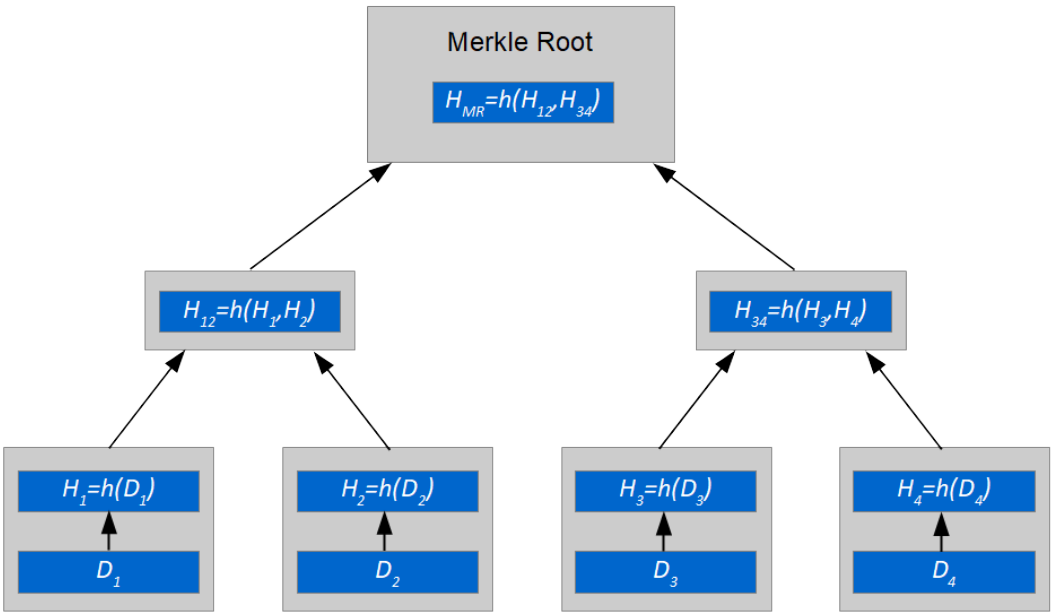


Bild 3.1.: Merkle Hash Tree

Um die Unverändertheit aller Datenpakete nachzuweisen, genügt es, den Merkle Root Hash zu verifizieren, denn eine Änderung auch nur in einem der Datenpakete hat aufgrund der Verkettung der Hash Werte zwangsläufig eine Änderung des Merkle-Root-Hashes zur Folge. Dieses Verfahren eignet sich zur Signatur einer Menge von Datenpaketen, die zeitgleich vorliegen.

3.1.2. Hash-Kette

Liegen Datenpakete nicht zum gleichen Zeitpunkt vor, sondern soll zu einem bestehenden Datenpaket ein weiteres Datenpaket zu einem späteren Zeitpunkt hinzugefügt werden, kann das Prinzip der Verkettung von Hash-Werten in Form einer verketteten Liste angewendet werden.
Jedes Listenelement besteht aus einem Datenpaket D_i und einem Header H_i. Der Header $H_i = h(H_{i-1}, D_i)$ berechnet sich als Hash über die Verkettung vom Header H_{i-1} des Listenvorgängers und dem Datenpaket D_i. Verdeutlicht ist dies in Bild 3.2. Der Hash-Wert über den gesamten letzten Listeneintrag (in diesem Beispiel H_4) kann dann zur Verifikation der Unverändertheit aller Listeneinträge genutzt werden. Jede Änderung in einem einzelnen Datenpaket bewirkt eine Änderung in allen nachfolgenden und damit auch des letzten Hashes.

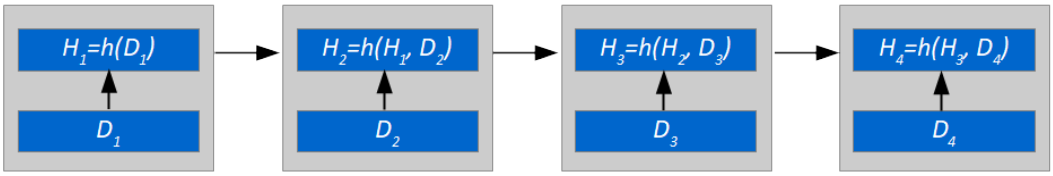


Bild 3.2.: Hash-Kette

3.1.3. Die Blockchain-Datenstruktur

Die Datenstruktur einer Blockchain ergibt sich nun aus der Kombination dieser beiden Verfahren wie in Bild 3.3 illustriert:

1. Mehrere Transaktionen (Datenpakete) werden in Form eines Merkle-Trees zu einem Blockdatenpaket mit dem Merkle Root Hash H_{MR} zusammengefasst.

2. Der Merkle-Root-Hash $H_{MR,i}$, ein Zeitstempel T_i und zwei weitere Elemente (der so genannte Nonce Wert N_i und der Schwierigkeitswert k_i, auf deren Bedeutung später genauer eingegangen wird) bilden gemeinsam die Headerinformationen des Blockes i.

3. Nun wird ein Block-Hashwert $H_{B,i} = h(H_{B,i-1}, H_{MR,i}, T_i, N_i, k_i)$ über den Block-Hash des Vorgängerblockes $H_{B,i-1}$ und die Headerinformationen des aktuellen Blockes i berechnet. Dieser Block-Hashwert $H_{B,i}$ signiert den Block-Header und das Block-Datenpaket, da ja der Merkle-Root-Hash teil des Headers ist.

Neue Blöcke werden also über den Block-Hash in Form einer Hash-Liste verkettet: Die Blockchain (-Datenstruktur).

3.2. Identifikation, Authentifikation und Authorisierung in der Blockchain

Zur Identifikation, Authentifikation und Authorisierung nutzt die Blockchain assymetrische Verschlüsselungsverfahren. Auch die Prinzipien dieses Konzeptes wurden bereits in den 1970er Jahren vorgestellt [DH76]. Ein Schlüsselpaar – bestehend aus einen öffentlichen (K_o) und einem privaten Schlüssel (K_p) – wird verwendet, um Daten zu ver- und entschlüsseln. Daten, die mit dem öffentlichen Schlüssel verschlüsselt werden, können nur mit dem privaten Schlüssel entschlüsselt werden und umgekehrt. Für ein Datenpaket D und eine asymmetrische Verschlüsselungsoperation $C(D,K)$ mit dem Schlüssel K auf dem Datenpaket D gilt also:

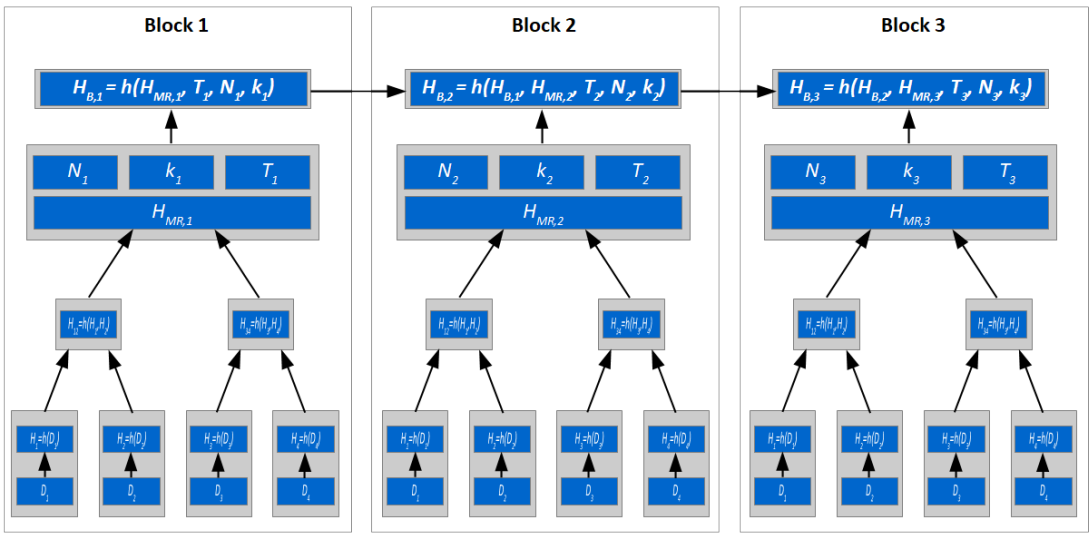


Bild 3.3.: Blockchain-Datenstruktur

$$D = C(C(D, K_o), K_p) \tag{3.2}$$

und

$$D = C(C(D, K_p), K_o). \tag{3.3}$$

Mittels der Operation (3.2) kann jeder, der den öffentlichen Schlüssel kennt, eine Nachricht verschlüsseln, die dann nur durch den Besitzer des privaten Schlüssels entschlüsselt werden kann. Mittels der Operation (3.3) kann jemand, der im Besitz eines privaten Schlüssels ist, eine Datenpaket signieren. Jedermann kann dann mittels des öffentlichen Schlüssels die Urheberschaft des Datenpaketes verifizieren. Für Details zu verschiedenen asymmetrischen kryptographischen Verfahren und deren Anwendung siehe z.B. [Jan16].
So dient der öffentliche Schlüssel in einer Blockchain zur Identifikation eines Adressaten und der private Schlüssel wird zur Signatur von Transaktionen und damit zur Authentifizierung des Absenders der Transaktion verwendet.

3.3. Vertrauen in der Blockchain

Die Blockchain Datenstruktur stellt sicher, dass einmal in der Blockchain abgelegte Daten nicht mehr verändert oder gelöscht werden können. Ferner wird über asymmetrische Verschlüsselungsverfahren Identifikation, Authentisierung und Authorisierung ermöglicht.

Jedoch kann jemand, der diese Datenstruktur kontrolliert, eine Manipulation vornehmen. Ändert er einen Datensatz und berechnet dann alle nachfolgenden Hashwerte neu, ergibt sich wieder eine integre Struktur. Für einen Dritten ist es nicht möglich, diese Manipulation festzustellen. Solange also nur eine Kopie der Blockchain Datenstruktur existiert, setzt die Integrität der Daten das Vertrauen in den Betreiber der Datenstruktur voraus, keine Manipulation vorgenommen zu haben.
Um dieses Problem zu lösen, sieht die Blockchain zwei Konzepte vor:

1. Die Kosten (der Aufwand) für eine mögliche Manipulation müssen groß sein. Dies wird durch das Proof-of-Work Konzept erreicht.

2. Kopien der Blockchain Datenstruktur werden auf Knoten in einem Peer-To-Peer Netzwerk[3] vorgehalten.

3.3.1. Proof of Work

Das Proof of Work Konzept sieht vor, dass das Hinzufügen eines gültigen Blockes zu einer Blockchain künstlich erschwert wird, in dem dazu ein rechenaufwändiger Algorithmus angewendet werden muss. Das von Nakamoto vorgeschlagene und in der Bitcoin Blockchain implementierte Verfahren nutzt hierzu auch wieder Eigenschaften der Hash-Berechnung.
Wie in Abschnitt 3.1.3 erläutert, muss jeder neue Block durch den Hash-Wert über den Header des Blocks (den Block-Hash) signiert werden. Der Header des Blocks enthält fixe Daten (die Hash-Signatur des vorhergehenden Blocks, den Merkle-Root-Hash des Blocks, den Zeitstempel des Blocks sowie den Schwierigkeitsgrad k) und ein weiteres Element, den so genannten Nonce N, der beliebig gewählt werden kann.
Selbstverständlich ändert sich der Block-Hash, wenn der Nonce variiert. Es kann daher eine Randbedingung an den Block-Hash gestellt werden, die durch nur durch zufällige Variation des Nonce erreichbar ist. In der Implementierung der Bitcoin Blockchain ist dies die Forderung, dass die ersten k Bits des Block-Hashs identisch 0 sein müssen. Bei einer Hash Länge von 256 Bit[4] gilt also:

$$H_{B,i}(N_i) = h(H_{B,i-1}, H_{MR,i}, T_i, N_i, k_i) \overset{!}{\leq} 2^{256-k}.$$

Je größer k, umso schwieriger ist es, durch Ausprobieren unter zufälliger Variation des Nonce N_i diese Forderung zu erfüllen.
Ein Block kann also der Blockchain nur hinzugefügt werden, nachdem das Rätsel „finde einen Nonce, für den der Block-Hash in den ersten k Bits identisch Null ist“

[3] Ein Peer-to-Peer Netzwerk ist ein Netzwerk, in dem grundsätzlich alle Knoten gleichberechtigt sind und jeder Knoten mit jedem anderen Knoten kommunizieren kann.
[4] Bitcoin verwendet als Hashing Verfahren SHA-256.

aufwändig gelöst wurde. Die Überprüfung, ob der Block-Hash korrekt ist und der Bedingung genügt, ist hingegen einfach und bedarf nur dem einmaligen Anwenden der Hash-Funktion.
In der Bitcoin Blockchain wird der Schwierigkeitsgrad regelmäßig in Abhängigkeit von der Gesamtrechenpower des Bitcoin Netzwerkes so angepasst, dass die durchschnittliche Zeit zur Berechnung eines Blockes mit der Gesamtrechenleistung im Bitcoin Netzwerk etwa 10 Minuten beträgt.
Beabsichtigt nun jemand eine Manipulation eines Datensatzes der Blockchain, muss er – wie bereits festgestellt – alle nachfolgenden Block-Hashes neu berechnen. Aufgrund des Proof of Work Konzeptes ist dies nun mit hohem Aufwand (und Kosten) verbunden.

3.3.2. Peer-to-Peer Blockchain-Netzwerk

Ein Peer-To-Peer Netzwerk ist „ein sich selbst organisierendes System gleichberechtigter, autonomer Einheiten (Peers), das vorzugsweise ohne Nutzung zentraler Dienste auf der Basis eines Rechnernetzes mit dem Ziel der gegenseitigen Nutzung von Ressourcen operiert“ [SW04].
Die Grundideen eines Peer-To-Peer Netzwerkes wurde bereits zu Zeiten des ARPANETS in den 1960er Jahren gelegt. Bekannt wurde das Konzept mit der Einführung des File Sharing Dienstes Napster 1999 und dem freien Napster Klon OpenNap [drs00]. Beiden Ansätzen ist aber gemein, dass sie entgegen der Definition auch über zentrale Komponenten verfügen.
Die Blockchain Architektur nutzt das Peer-to-Peer Paradigma in seiner reinen Form:

- Jeder Knoten im Peer-to-Peer Netzwerk hält eine Kopie der Blockchain-Datenstruktur.
- Jeder Knoten ist gleichberechtigt.
- Jeder Knoten kann prinzipiell mit jedem anderen Knoten zum Zwecke des Abgleichs Nachrichten austauschen.

Das verwendete Peer-to-Peer Protokoll ist dabei fehlertolerant. Das Funktionalität des Gesamtsystems ist nicht davon abhängig, dass einzelne Nachrichten ankommen oder dass einzelne Knoten verfügbar sind.
Dadurch, dass nun auf jedem Knoten eine Kopie der Blockchain liegt, ist es einem Einzelnen nicht mehr möglich, unbemerkt Manipulationen vorzunehmen. Zwar kann er seine lokale Kopie der Blockchain manipulieren und ihre scheinbare Integrität nachweisen. Sie wird jedoch zwangsläufig von allen anderen Kopien abweichen und damit als nicht integer identifizierbar sein.

Die Peer-to-Peer Architektur der Blockchain vermag damit grundsätzlich, das Vertrauen in die Integrität der Datenstruktur ohne einen vertrauenswürdigen Mittelsmann herzustellen, wenn es gelingt, dass mindestens die Mehrheit der Knoten im Netzwerk den gleichen integren Datenbestand hält.
Auch, wenn zu einem Zeitpunkt t_0 alle Knoten einen identischen Datenbestand D_0 haben, kann es allerdings passieren, dass verschiedene Knoten unterschiedliche neue Daten dem Datenbestand hinzufügen und der Datenbestand auf den einzelnen Knoten dann nicht mehr zwangsläufig identisch ist. Es bedarf daher eines Verfahrens, um Konsens über den „wahren“ Datenbestand zu erzielen. Dies wird mit dem so genannten Consensus-Algorithmus erreicht.

3.3.3. Consensus Algorithmus in der Blockchain

Der Consensus Algorithmus verbindet die beiden Konzepte Proof-of-Work und Peer-to-Peer Architektur, um Manipulationen zu identifizieren und zu verhindern und einen eindeutigen konsistenten Datenbestand zu gewährleisten. Er folgt dabei den Grundsätzen:

1. Der wahre Datenbestand wird immer durch die längste gültige Blockchain gebildet.

2. Gültig ist eine Blockchain, die von der Mehrheit der Knoten als gültig validiert wurde.

3. Ein Knoten wird incentiviert, wenn er neue Transaktionen validiert (die Regeln hierzu sind jeweils anwendungsspezifisch) und gültige und autorisierte Transaktionen als neuen Block der wahren Blockchain hinzufügt. Hierzu muss er den Proof-of-Work erfolgreich leisten.

4. Jeder Knoten kontrolliert andere Knoten.

Hierzu hat jeder Knoten prinzipiell (durch Nachrichtenaustausch mit anderen Knoten) Zugriff

- auf Datensätze (Transktionen), die der Blockchain hinzugefügt werden sollen und

- auf Blöcke, die von anderen Knoten der Blockchain hinzugefügt wurden.

Die Incentivierung ist notwendig, da die Beteiligung eines Knotens am Consensus Verfahren mit Aufwand und Kosten verbunden ist. Im Fall der Bitcoin Blockchain erfolgt die Incentivierung durch Ausschüttung von Bitcoins.
Ein Knoten folgt in einer Schleife dem folgenden Algorithmus:

1. Wenn ein neuer Block gemeldet wird: validiere den Block. Die Validierung besteht in
   - der Überprüfung der Folge (ist der Block ein Folgeblock der aktuell gültigen Blockchain?),
   - der Überprüfung des Block-Hashes (ist der Block-Hash korrekt und wurde der Proof-of-Work erfolgreich gelöst?) und
   - der Überprüfung der Transaktionen im Block auf Gültigkeit (inkl. Prüfung des Merkle-Root-Hashs).
2. Wenn der Block als gültig validiert wurde, akzeptiere ihn als neuen Block der (lokalen) Blockchain.
3. Ungültige Blöcke werden verworfen (und andere Knoten per Nachricht darüber informiert). Transaktionen aus verworfenen Blöcken werden wieder dem Pool neuer Transaktionen zugefügt.
4. Versuche selbst, den nächsten gültigen Block zu generieren (und dafür incentiviert zu werden). Löse dafür den Proof-of-Work. Informiere die anderen Knoten, sobald ein gültiger Block gefunden wurde.
5. Wenn eine Nachricht kommt, dass ein Block als ungültig identifiziert wurde, der Teil der lokalen gültigen Blockchain ist, dann setze die lokale Blockchain zurück bis auf den Vorgänger dieses ungültigen Blockes.

Mit diesem auch als Mining bezeichneten Vorgehen wird folgendes erreicht:

1. Nur gültige Transaktionen werden (letzlich) der Blockchain hinzugefügt.
2. Alle Knoten sind grundsätzlich gleichberechtigt, neue Blöcke zu generieren. Es obliegt dem Zufall und der Rechenleistung eines Knotens, welcher Knoten den Proof-of-Work als erstes löst.
3. Nur als gültig validierte Blöcke werden von anderen Knoten akzeptiert und ihrer lokalen Blockchain angefügt.

Das Consensus-Verfahren dient dazu sicherzustellen, dass bei n verteilten Knoten im Blockchain Netzwerk die „wahre“ Information in der Blockchain gespeichert wird, auch wenn einzelne Knoten nicht vertrauenswürdig sondern manipulativ agieren. Ein Mehrheitsentscheid sichert einen konsistenten und wahren Datenbestand.
Das hier zu lösende grundsätzliche Problem wurde bereits 1982 als „das Problem der byzantinischen Generäle“ formuliert [LSM82]. n Generäle können nur über Boten Nachrichten austauschen, müssen aber zu einer gemeinsamen Entscheidung kommen.

Nicht alle Generäle sind loyal. Lamport et al. zeigen in ihrem Artikel, dass ein Algorithmus existiert, der das Problem sicher löst, wenn mehr als $2/3$ der Generäle loyal sind. Als byzantin fehlertolerant werden verteilte Systeme bezeichnet, die globale Zuverlässigkeit des Systems garantieren, auch wenn einige Komponenten unzuverlässig sind oder sogar böswillig manipulativ agieren. Ein Algorithmus auf Basis der Replikation von Zustandsautomaten wurde 1999 von Castro und Liskov vorgestellt und 2002 veröffentlicht [CL02]. Auch die Blockchain kann als Zustandsautomat interpretiert werden, und Miller und LaViola konnten beweisen, dass beim Bitcoin Proof-of-Work Consensus Verfahren in der Tat bereits die einfache vertrauenswürdige Mehrheit (in diesem Fall der Rechenleistung im Bitcoin Netzwerk) zu einem wahren Konsens führt [MJ14].
Das Proof-of-Work Verfahren ist damit ein zentrales Element für das Vertrauen in die Blockchain ohne Mittelsmann, indem es byzantine Fehlertoleranz sicherstellt.
Es ist möglich, das temporär aufgrund der asynchronen Kommunikation zwischen den Knoten mehrere gültige Blockchains gleicher Länge existieren („Race Condition“). Zu einem gewissen Zeitpunkt wird aber eine dieser Blockchains länger sein als die anderen und sich als „die wahre“ Blockchain durchsetzen. Letztlich wird sich also bei der Mehrheit der Knoten die gleiche gültige Blockchain einstellen („eventual synchrony“, [DLS88]).
Nachteilig ist, das während einer Race Condition die differierenden Blöcke (und die darin enthaltenen Transaktionen) nicht als final bestätigt betrachtet werden können. Erst, wenn sich wieder nur eine wahre Blockchain durchgesetzt hat, ist globale Konsistenz gewährleistet. Problematisch hierbei ist zum einen die zeitliche Verzögerung bis zur Auflösung der Race Condition, zum anderen der Umstand, dass einzelne Beteiligte nicht wissen können, ob gerade eine Race Condition vorliegt. Für die Bitcoin Blockchain hat sich als Erfahrungswert herausgestellt, dass Race Conditions mit hoher Wahrscheinlichkeit nach spätestens sechs Blöcken aufgelöst werden.

3.4. Blockchain Varianten

Das Consensus Verfahren nach Abschnitt 3.3.3 ist das zentrale Element der Blockchain, um Vertrauen in einen Mittelsmann durch Vertrauen in die Mehrheit der Teilnehmer ersetzen zu können.
Dennoch gibt es Kritik an diesem Verfahren. Der Energieverbrauch im Bitcoin Netzwerk ist enorm [Giu+17]. Mining lohnt sich nicht mehr für „kleine“ Knoten, so dass sich mehr und mehr die Entscheidungsfindung auf immer weniger Knoten mit immer mehr Rechenleistung beschränkt [TS16]. Dies widerspricht dem Grundgedanken der freien dezentralen Entscheidungsfindung. Es existieren daher auch andere Konsensus-

Verfahren, von denen einige im folgenden kurz skizziert werden.

3.4.1. Alternative Consensus Verfahren

Proof of Stake

Proof of Stake [KN18] basiert auf den Konzept der Coinage. Die Coinage ist definiert als das Produkt aus Geldwert einer digitalen Münze multipliziert mit der „Haltezeit“ (Zeit, seitdem diese Münze das letzte Mal in einer Transaktion verwendet wurde). Die Generierung eines Blockes basiert weiterhin auf der Lösung eines „Hash Puzzles“. Allerdings sieht der Algorithmus vor, dass die Lösung dieses Rätsels umso einfacher wird, je mehr Coinage der Miner einbringt. Der Incentive besteht darin, dass der erfolgreiche Miner eine Vergütung in Abhängigkeit dieser eingebrachten Coinage erhält. Nach dem erfolgreichen Minen eines Blockes ist dann allerdings die Coinage verbraucht (und steigt erst wieder mit fortscheitender Zeit). So wird gewährleistet, dass deutlich stärker der Zufall und nicht die Rechenleistung darüber entscheidet, wer einen Block abschließt.
Coinage ist auch das Kriterium für die Entscheidung über die „wahre“ Blockchain im Falle von sich bildenden parallelen Ketten. Nicht die längste Kette, sondern die Kette mit dem höchsten Wert an Coinage in den Transkaktionen gewinnt.

Proof of Authority

Proof of Authority [Net17] ist eine Variante des Proof of Stake, bei der alleine die Identität eines Teilnehmers ihn zur Validierung von Blöcken berechtigt. Dies setzt einen externen standardisierten Prozess voraus, in dem einem Teilnehmer zunächst eine Identität auf Basis seiner Reputation zugesprochen wird. Die Kosten zur Erlangung einer Identität müssen angemessen hoch sein, so dass der Verlust bei missbräuchlicher Nutzung entsprechend schmerzhaft ist. Ferner muss sichergestellt werden, dass ein Teilnehmer nur eine Identität inne haben kann.

Proof of Activity

Proof of Activity [Ben+14b; Ben+14a] nutzt grundsätzlich auch den Proof-of-Work Algorithmus. Allerdings werden jeweils zusätzlich N Teilnehmer am Netzwerk zufällig ausgewählt, die sich an der Berechnung eines Blockes beteiligen müssen. Die Wahrscheinlichkeit für die Auswahl ist abhängig von der Anzahl an Münzen, die der Teilnehmer hält. Gleichzeitig muss er aber auch online sein. Vorteile, die die Initiatoren dieses Verfahrens anführen, sind eine höhere Motivation für alle Teilnehmer, sich am Blockchain Algorithmus zu beteiligen und eine geringere Abhängigkeit von der Rechenleistung eines Einzelnen. Der Zufallsprozess der Auswahl der Teilnehmer sorgt

für eine geringere Abhängigkeit von der Rechenleistung, und der Schwierigkeitsgrad des Hash Puzzles kann geringer gewählt werden, was den Aufwand und damit den Energieverbrauch sinken lässt.

Proof of Burn

Die Wahrscheinlichkeit, im Proof-of-Work Consensus Verfahren einen Block zu signieren, steigt mit der Rechenleistung. Mit entsprechender Investition in Hardware und Energiekosten kann ein Miner diese Wahrscheinlichkeit also steigern. Proof-of-Burn [P4T14] zielt darauf ab, die Kosten in Hardware und Energie zu ersetzen durch Invest in Form von Kryptowährung. Kryptomünzen, die auf einen speziellen Account übertragen werden, gelten als „verbrannt“ und sind dauerhaft unbrauchbar. Der Gegenwert dieser verbrannten Münzen kann dazu genutzt werden, die Wahrscheinlichkeit für das erfolgreiche Signieren eines Blockes zu steigern. Ähnlich wie der Invest in Hardware über der Zeit abgeschrieben werden muss, unterliegen auch die verbrannten Münzen einem Wertverlust im Zeitverlauf.

Proof of Luck

Ein weiteres vorgeschlagenes Consesus Protokoll ist Proof-of-Luck [Mil+16]. Hierbei ist es gänzlich dem Zufall überlassen, welcher Miner den aktuellen Block signiert. Allerdings ist die Berechnung gekoppelt an ein Trusted Execution Environment (TEE), eine hardwarebasierte vertrauenswürdige Laufzeitumgebung. In der TEE wird eine Zufallszahl ermittelt, in Abhängigkeit von der Zufallszahl eine bestimmte Zeit gewartet, dann der Block signiert und an die anderen Knoten kommuniziert. Je höher die Zufallszahl („Luck“), umso kürzer die Wartezeit und daher umso wahrscheinlicher, dass der eigene Block der Blockchain hinzugefügt wird.
Entscheidungskriterium für die „wahre“ Blockchain im Falle von Verzweigungen ist, die Blockchain zu wählen, bei der die Summe der Zufallswerte der Blöcke maximal wird („the luckiest“).
Durch die Verwendung von TEEs wird sichergestellt, dass jeder Knoten im Netzwerk gleichberechtigt ist. Jeder Knoten kann nur ein TEE einsetzen. Ein manipulativer Angriff mit einer gewissen Erfolgswahrscheinlichkeit setzt die Verfügbarkeit über die Mehrheit der TEEs voraus.

3.4.2. Blockchains mit eingeschränktem Nutzerkreis

Es gibt Blockchain Anwendungsfälle mit eingeschränktem Nutzerkreis. Für die weitere Betrachtung ist eine einfache Differenzierung in öffentliche, private und Konsortium Blockchains angemessen. Zugangsbeschränkte private und Konsortium Blockchains

werden auch als permissioned BC, jedermann zugängliche Blockchains wie Bitcoin als permissionless BC bezeichnet.

Öffentliche Blockchains

Nur eine öffentliche Blockchain basierend auf Open Source Software wird den Prinzipien der Dezentralisierung, dem Vertrauen nur in die Mehrheit aller Beteiligten und nicht in Einzelne sowie der Transparenz voll gerecht:

1. Jeder kann den Code lesen und prüfen.
2. Jeder kann alle Transaktionen lesen und prüfen.
3. Jeder kann sich am Consensus Prozess beteiligen.

Private Blockchains

Beschränkt sich der Nutzerkreis auf nur eine Partei, spricht man von einer privaten Blockchain. Bei einer privaten Blockchain steht sicher die Funktion der Unveränderbarkeit der Daten durch die Blockchain-Datenstruktur im Vordergrund. Das Thema „Vertrauen“ ist hier von untergeordneter Bedeutung. Im Hinblick auf die Verfügbarkeit ist weiterhin der Einsatzes mehrerer Knoten in einem Netzwerk sinnvoll. Auch hier ist ein Consensus Protokoll notwendig, um im Sinne der „eventual synchrony“ einen konsistenten Status in einem ausfallbehafteten asynchronen System zu gewährleisten. Der Consensus Algorithmus kann aber in einer privaten Blockchain deutlich vereinfacht werden, da er nur tolerant gegebenüber technischen Fehlern und Ausfällen, nicht aber gegenüber manipulativem Verhalten sein muss. Denkbar ist z.B. ein Proof-of-Luck Verfahren, das keine TEEs erfordert oder ein so genanntes Atomic Broadcast Verfahren wie Paxos [Lam01].

Konsortium Blockchains

Eine Konsortium Blockchain kann sinnvoll sein, wenn mehrere Parteien Daten gemeinsam nutzen, diese Daten aber nicht zentral einer der Parteien oder einem Dritten anvertrauen wollen. Gemeinsame Nutzung der Blockchain gewährleistet Transparenz für alle Beteiligten. Ein Anwendungsszenario ist beispielsweise das Logging des Lebenszyklus eines Produktes in einer Lieferkette vom Produzenten über verschiedenste Stationen bis zum Endverbraucher.
Die Konsortium Blockchain läuft in einem Netzwerk mit (vom Konsortium) ausgewählten und daher schon mit einem gewissen „Vorvertrauen“ ausgestatteten Knoten. Das Prinzip der Dezentralisierung wird nur noch eingeschränkt umgesetzt. Auch in diesem Kontext erscheint es sinnvoll, als Consensus Protokoll eine der Alternativen

zum kostenintensiven Proof-of-Work einzusetzen, z.B. Proof-of-Stake. Einen Überblick über Consensus-Protokolle, die in permissioned BCs aktuell zum Einsatz kommen, sind in [CV17] zusammengetragen.

3.5. Smart Contracts

Der Begriff Smart Contract wurde 1994 von Szabo geprägt als „ein computergesteuertes Transaktionsprotokoll, das die Bedingungen eines Vertrags ausführt“ mit dem Ziel „der Erfüllung gemeinsamer Vertragsbedingungen, der Minimierung von sowohl bösartigen als auch zufälligen Ausnahmen und der Minimierung des Bedarfs an vertrauenswürdigen Intermediären“[Sza94]. In Folge führt er dies detaillierter aus, erläutert an verschiedenen Anwendungsbeispielen und diskutiert die Möglichkeit, Smart Contracts über „Mutally Confidential Computation“ abzubilden [Sza97].
Primäres Ziel der Bitcoin Blockchain ist die Verwaltung von Bitcoin Transaktionen. Blockchain Technologien bilden jedoch auch eine ideale Basis, um Smart Contracts umzusetzen. Alle grundlegenden Voraussetzungen zum Transfer von Eigentum und/oder Krypto-Geldwerten ohne einen vertrauenswürdigen Mittelsmann sind gegeben. So wurden verschiedene Smart Contracts durch Skripte im Bitcoin Protocol implementiert [Atz+18].

3.5.1. Smart Contracts in der Ethereum Blockchain

Mit Ethereum steht eine Blockchain Technologie zur Verfügung, die nicht nur Transaktionsdaten in der Blockchain speichert, sondern die es jedermann ermöglicht, Smart Contracts als ausführbare Programme in der Ethereum Blockchain anzulegen, zu verwalten und in einer Ethereum Virtual Machine (EVM) auszuführen:

> Ethereum does this by building what is essentially the ultimate abstract foundational layer: a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats and state transition functions[But+13].

Eine Transaktion in der Ethereum Blockchain ist damit nicht auf den Transfer von geldwerten Tokens („Ether“, die Kryptowährung von Ethereum) beschränkt. Mittels einer Transaktion kann der Code eines Smart Contracts unveränderbar in der Blockchain hinterlegt oder eine Funktion in einem Smart Contract aufgerufen werden. Allen Transaktionen gemein ist, dass sie den Zustand der Blockchain ändern.
Ethereum nutzt aktuell Proof-of-Work als Consensus Verfahren. Eine Umstellung auf Proof-of-Stake ist vorgesehen [But16a].

Der Smart Contract wird als EVM Bytecode in der Blockchain abgespeichert [Woo18]. Er erhält eine eigene Adresse (Ethereum unterscheidet zwischen Contract Accounts und Nutzer Accounts). Ein Vertrag[5] kann daher Transaktionen von anderen Nutzern und Verträgen empfangen, und er kann Transaktionen an andere Verträge oder Nutzer senden.
Der Vertrag kann als Zustandsautomat interpretiert werden, dessen Status persistent in der Blockchain gespeichert ist. Ethereum erweitert damit das Prinzip der byzantin fehlertoleranten Zustandsautomaten [Sch90], indem es die quasi parallele Ausführung einer Vielzahl dieser Zustandsautomaten in Form von Smart Contracts erlaubt, die von jedermann implementiert werden können.
Der EVM Bytcode eines einmal in der Ethereum Blockchain abgelegten Vertrages kann nicht mehr geändert oder gelöscht werden. Die einzige Ausnahme hiervon besteht darin, das im Vertrag selbst eine Funktion zur Löschung des Vertrages implementiert wird, wobei dies wiederum an Bedingungen geknüpft werden kann, z.B., dass nur der Eigentümer des Vertrages den Vertrag löschen darf.
Die Aktivierung des Vertrages erfolgt, indem eine Transaktion (von einem Nutzer oder einem anderen Vertrag) an die Adresse des Vertrags gesendet wird. Jede solche Transaktion überträgt einen geldwerten Betrag ≥ 0 in Ether an den Vertrag und kann einen Funktionsaufruf mit Eingabeparametern enthalten. Mit Empfang der Transaktion

- sammelt der Vertrag das gesendete Geld ein,
- führt die aufgerufene Funktion mit den Eingabeparametern aus,
- aktualisiert hierbei ggf. den internen Status und/oder initiiert Transaktionen und
- kann Events generieren, um über Statusänderungen des Vertrages zu informieren.

Nachfolgend wird ein einfaches Beispiel betrachtet. In einer Ethereum-Blockchain seien durch einen Smart Contract (ZV-Vertrag) Regeln implementiert, um Zertifikate zu verwalten, die das Eigentum an bestimmten Gütern dokumentieren. Zur Übertragung eines Zertifikates stelle dieser Vertrag die Funktion `ZT(Zertifikat, Empfänger)` zur Verfügung.
Alice möchte Zertifikat A für 5 Ether an Bob verkaufen. Sie implementiert dazu den Smart Contract nach Listing 3.1. Tabelle 3.1 listet den Zustand von Guthaben und Zertifikatsbesitz für die Accounts von Alice, Bob und Vertrag in Abhängigkeit von den durchgeführten Transaktionen.
Alice verfügt zu Beginn über ein Guthaben von 10 Ether und ist im Besitz von Zertifikat A. Bob hat ein Initialguthaben von 25 Ether und besitzt noch kein Zertifikat.

[5] Vertrag wird im Folgenden in dieser Arbeit synonym für Smart Contract verwendet.

Listing 3.1: Pseudocode eines einfachen Smart Contracts

```
contract VerkaufeZertifikat {

  function verkaufeDochNicht() {
        pruefeBedingung:       Absender == Alice;
        pruefeBedingung:       Zertifikat gehört diesem Vertrag;
        initiiereTransaktion: Übertrage Zertifikat
                               von diesem Vertrag auf Alice;
  }

  function bezahle() {
        pruefeBedingung:       Absender == Bob;
        pruefeBedingung:       Zertifikat gehört diesem Vertrag;
        pruefeBedingung:       Transaktion beinhaltet 5 Ether;

        fuehreAus:             Übertrage Ether
                               aus Transaktion auf Vertrag;
        initiiereTransaktion: Übertrage 5 Ether
                               von Vertrag auf Alice;
        initiiereTransaktion: Übertrage Zertikat
                               von diesem Vertrag auf Bob;
  }
}
```

Alice initiiert den Verkauf, indem sie ihr Zertifikat A auf den Vertrag überträgt. Diese Funktionalität muss nicht im Vertrag implementiert werden, sie steht über die Zertifikatsverwaltung `ZT()` zur Verfügung. Implementiert werden muss hingegen eine Funktion `verkaufeDochNicht()`, die Alice die Möglichkeit gibt, ihr Zertifikat zurück zu übertragen, sollte Bob nicht zahlen.
Bob triggert dann die Funktion `bezahle()` durch eine entsprechende Transaktion. In dieser Funktion werden zunächst die Vertragsbedingungen geprüft: Ist Bob der Absender der Transaktion, ist der Vertrag im Besitz des Zertifikates und hat Bob mit der Transaktion 5 Ether an den Vertrag übertragen? Nur, wenn diese Bedingungen erfüllt sind, initiiert der Vertrag die notwendigen Transaktionen zur Abwicklung des Verkaufs. 5 Ether werden vom Vertrag an Alice gesendet und das Zertifikat wird an Bob übertragen.

Die Ausführung einer Vertragsfunktion ist transaktionssicher. Wird die Funktion nicht vollständig durchlaufen sondern z.B. aufgrund der Prüfung einer Bedingung die Ausführung abgebrochen, erfolgt keine Zustandsänderung.
Jeder Vertrag wird auf jedem Knoten der Blockchain ausgeführt. Jede Transaktion und jede Zustandsänderung wird unveränderbar gespeichert und ist, ebenso wie der Vertragscode selbst, für jedermann jederzeit prüfbar. Die Ausführung erfolgt im Order-Execute Modell [And+18]:

Tabelle 3.1.: Transaktionen zur Ausführung des Verkaufs des Zertifikates A für 5 Ether von Alice an Bob sowie die jeweiligen Zustände (Guthaben/Zertifikate) der Accounts in der Blockchain.

Transaktion				Zustand		
von	an	Funktion	Betrag	Alice	Bob	Vertrag
				10/A	25/-	0/-
Alice	ZV-Vertrag	`ZT(A,Vertrag)`	0	10/-	25/-	0/A
Bob	Vertrag	`bezahle()`	5	10/-	20/-	5/A
Vertrag	Alice	-	5	15/-	20/-	0/A
Vertrag	ZV-Vertrag	`ZT(A, Bob)`	0	15/-	20/A	0/-

1. Transaktionen, die an einen Smart Contract adressiert sind, werden zunächst wie jede andere Transkation in einen Block aufgenommen und so global als gültig verifiziert (Order).

2. Verifizierte Vertragstransaktionen werden auf allen Knoten ausgeführt (Execute).

Die Ausführung von Code in der Ethereum Blockchain muss vergütet werden. Dies ist vor allem eine Schutzfunktion, um übermäßigen Verbrauch der Rechenressourcen entgegen zu wirken und Denial of Service (DoS) Angriffe durch „bösartige“ Smart Contracts zu verhindern. Die Kosten richten sich nach der Anzahl und der Komplexität der ausgeführten Bytecode Operationen sowie des beanspruchten Speicherplatzes für Zustandsdaten. Sie werden in „gas“ angegeben und dann in Ether umgerechnet, wobei der Ether-gas Kurs Schwankungen unterliegen kann.
Einfache Transaktionen an den Vertrag, die nur z.B. den Zustand einer Variablen abfragen, aber den Zustand des Vertrages nicht ändern und keine weiteren Transaktionen auslösen, sind in der Ethereum Blockchain kostenfrei.

3.5.2. Smart Contracts in permissioned Blockchains

Das Order-Execute Modell für Smart Contracts hat einschränkende Randbedingungen zur Folge [And+18]:

- Alle Transaktionen müssen auf allen Knoten sequentiell ausgeführt werden. Dies limitiert den Durchsatz.

- Jede Einzeloperation muss deterministisch sein, ansonsten würde durch die Ausführung der gleichen Transaktion auf unterschiedlichen Knoten ein unterschiedlicher Zustand und damit kein Konsens erreicht. Ethereum stellt dies sicher, in

dem Smart Contracts in der spezielle Programmiersprache Solidity [Sol18] entwickelt werden, die nur deterministische Operationen erlaubt.

- Es muss sichergestellt sein, dass Smart Contracts in definierter Zeit terminieren. Endlos-Schleifen würden die komplette Blockchain aushebeln und wären ein einfacher DoS Angriffspunkt. Ethereum löst dies durch Kosten für die Ausführung, was wiederum die Abbildung einer Kryptowährung in der Blockchain voraussetzt. Ferner werden die maximalen Kosten eines Smart Contracts in Ethereum limitiert. Code beliebiger Komplexität ist daher nicht umsetzbar.

- Da jeder Smart Contract auf jedem Knoten ausgeführt wird, ist der Code für alle Teilnehmer transparent. In Ethereum ist dies explizit gewünscht, eine Anforderung in permissioned BCs kann jedoch sein, dass nicht jeder Code von jedem Teilnehmer einsehbar ist, sondern nur auf einer Untermenge der BC Knoten ausgeführt wird.

Aus diesem Grund haben sich verschiedene permissioned BC Technologien entwickelt [CV17]. Ein flexibles Framework ist Hyperledger, eine Open-Source-Kooperation, die „entwickelt wurde, um branchenübergreifende Blockchain-Technologien voranzutreiben“ [Hyperledger]. Ein Hyperledger Projekt ist Hyperledger Fabric [And+18]. Fabric folgt nicht dem Order-Execute, sondern dem Execute-Order-Validate Modell:

1. An Smart Contracts (hier Chaincode genannt) gerichtete Transaktionen werden von so genannten Endorsement Nodes (die je Chaincode konfigurierbar sind und so eine gewisse Vertraulichkeit hinsichtlich des Codes gewährleisten) ausgeführt.

2. Die Ergebnisse der Ausführung (in Form von Zuständen) werden dann an alle Knoten zur Integration in einen Block der BC gesendet. Das hierbei zur Anwendung kommende Consensus Protokoll ist variabel.

3. Die Validierung der Daten erfolgt erst abschließend („post-consensus validation“).

Die Autoren reklamieren, dass dadurch

- Smart Contracts in beliebiger Programmiersprache erstellt werden können,

- Performanzprobleme gelöst werden, da Chaincode nur auf ausgewählten (Smart Contract spezifischen) Endorsement Knoten ausgeführt werden muss,

- Auswirkungen von nicht-deterministischem Verhalten begrenzt wird,

- der Chaincode vertraulich behandelt werden kann und

- das Consensus Protokoll entkoppelt und damit individuell wählbar ist.

3.6. Zusammenfassung

Die Blockchain Architektur verbindet in einem Peer-to-Peer Netz gleichberechtigte Knoten, von denen jeder eine lokal Kopie der Blockchain Datenstruktur hält. In dieser Datenstruktur werden Transaktionsdaten unveränderbar und jederzeit überprüfbar abgelegt. Jeweils mehrere Datensätze werden hierzu in Form eines Merkle Hash Trees zu einem Block zusammengefasst, und über der Zeit werden alle Blöcke in Form einer Hash Kette verbunden. Der globale Abgleich der lokalen Daten im Peer-To-Peer Netzwerk erfolgt über Consensus Verfahren, die in mehreren Ausprägungen zur Verfügung stehen. Diese Verfahren sind nicht nur robust gegenüber technischen Fehlern (z.B. dem Ausfall einzelner Knoten oder einzelner Kommunikationsverbindungen), sondern können auch bewusst böswilliges manipulatives Verhalten einzelner Knoten identifizieren und abwehren. Damit ist es möglich, rein systembedingt Vertrauen in die Daten der Blockchain sicherzustellen, ohne auf einen vertrauenswürdigen Mittelsmann angewiesen zu sein.
Zur Identifikation, Authentifikation, und Authorisierung von Nutzern der Blockchain werden assymetrische Verschlüsselungsverfahren verwendet.
Dem Grundgedanken eines verteilten, offenen und damit transparenten Systems folgendend sind Blockchains zunächst öffentlich und jedermann zugänglich. Die Nutzung der Technologie kann jedoch auch in geschlossenen privaten oder konsortialen Blockchains mit eingeschränktem Nutzerkreis, so genannten permissioned Blockchains, sinnvoll sein.
Eine wesentliche Erweiterung der Möglichkeiten der Blockchains stellen Smart Contracts dar. Ein Smart Contract bildet Regelungen zwischen Nutzern der Blockchain in Form eines Programmcodes ab, der verbindlich und unveränderbar in der Blockchain abgelegt wird. Auch die Ausführung dieses Vertrages und damit die Anwendung der implementierten Regelungen erfolgt protokolliert, überprüfbar und transaktionssicher in der Blockchain.

4. Einsatzmöglichkeiten von Blockchains für das Internet der Dinge

Brody und Pureswaran fordern für ein zukunftsfähiges IoT Unabhängigkeit von zentralen Kommunikationsservern und Datenspeichern ebenso wie von der Notwendigkeit, in einzelne Personen oder Organisationen vertrauen zu müssen [BP14]. Konsequenterweise stellen sie als Anspruch an eine solche IoT Lösung, dass sie

- vertrauenslosen[1] Peer-to-Peer Nachrichtenaustausch,
- sicheren verteilten Datenaustausch und
- eine robuste und skalierbare Form der Koordination der IoT Endgeräte

unterstützt.
Global betrachtet ist das IoT ein Netz von Akteuren (Dingen und Nutzern), die potentiell alle miteinander Transaktionen ausführen. Keineswegs können alle Akteure im IoT als vertrauenswürdig betrachtet werden. Es ist daher ein geeignetes Verfahren notwendig, um diese Transaktionen zu validieren. Genau dies ist es, was Blockchain Techologien leisten können. Zunächst werden grundsätzliche Einsatzmöglichkeiten betrachtet, bevor einige konkrete Anwendungsbeispiele vorgestellt werden.

4.1. Grundsätzliche Einsatzmöglichkeiten

Verwaltung von Rechten

Eine BC kann dazu verwendet werden, Beziehungen zwischen Nutzern und IoT Geräten und Zugriffsrechte auf IoT Geräte zu verwalten: Wer hält das Eigentum an einem IoT Gerät? Wer darf auf welche Daten zugreifen? Hierbei können zusätzliche Kriterien wie z.B. zeitliche Einschränkungen, aber auch räumliche Nähe oder andere Informationen, die in der BC verbindlich und validiert abgelegt sind, Berücksichtigung finden und so eine dynamische Rechteverwaltung ermöglichen.

[1] im Sinne von Vertrauen ohne Mittelsmann

Führen von Herkunftsnachweisen, Asset Tracking

Die Historie eines Wertes, dessen Statusänderungen über Transaktionen in der BC gelogged werden, kann jederzeit überprüft und zurückverfolgt werden. Dies ist u.a. für das Management von Lieferketten relevant. So können beispielsweise verbaute Ersatzteile bis zu den für die Herstellung benutzen Rohprodukten verfiziert werden. Andere Anwendungsfälle sind die Zusammensetzung von Medikamenten oder der Echtheitsnachweis von Luxusprodukten. Assets können auch immaterielle Werte wie Informationen oder Wissen sein. BC Technologien sind hier universell einsetzbar.

Selbstverwaltung von IoT Geräten

Smart Contracts erlauben die Implementierung von Regelungen, die IoT Geräten eine von der Herstellerinfrastruktur unabhängige Selbstverwaltung gestatten. Dies kann reichen von dezentralen Software-Updates bis hin zu eigenständigem Handel mit Ressourcen, z.B. der Bereitstellung von Rechenleistung oder Speicherplatz.

Verwaltung von Regeln für Transaktionen zwischen IoT Geräten

In Smart Contracts können Regeln abgebildet werden, die die direkte selbstorganisierende Abwicklung von Transaktionen zwischen IoT Geräten ermöglicht: Ein Gerät kann von einem anderen Gerät Ressourcen oder Informationen anfordern, z.B. Speicherplatz. Die Transaktion kann in Form eines Tauschgeschäftes laufen oder über die Kryptowährung der BC oder einer anderen Form von Token vergütet werden. Solange die Transaktionskosten in der BC gering sind, lassen sich Micropayment Geschäfte realisieren, die auf herkömmlichen Wegen nie wirtschaftlich darstellbar wären.

Mehrheitsentscheidungen über die Vertrauenswürdigkeit einzelner IoT Geräte

Wenn ein IoT Gerät nicht nur auf die BC für die Abwicklung von Transaktionen zugreift, sondern selbst ein Teil der BC ist und am Consensus Prozess teilnimmt, werden über den Consensus Prozess nicht vertrauenswürdige IoT Geräte identifiziert. Es ist prinzipiell möglich, solche byzantinen IoT Geräte zu isolieren und vom Netzwerk auszuschließen oder mit Kosten für manipulative Transaktionen zu belegen.

Mehrheitsentscheidungen über die Vertrauenswürdigkeit von Daten

Wenn ein Algorithmus zur Validierung von Daten existiert, wird das Vertrauen in diese Daten automatisch über den Consensus Algorithmus abgesichert. Nur Daten, die den Validierungs- und Consensusprozess der BC bestehen, werden der BC hinzugefügt.

Durchführung automatisierter Geräteprüfungen

In einem Smart Contract können Regeln für die Überprüfung eines IoT Gerätes hinterlegt werden, z.B. die Abfrage von Statuswerten. Diese Überprüfung kann automatisiert über Transaktionen ablaufen, das Protokoll der Transaktionen in der BC dient automatisch als vertrauenswürdiges Prüfprotokoll.

Kontrolle über die Verwendung der Daten

Werden Daten grundsätzlich nur über BC Transaktionen ausgetauscht, ist jederzeit nachvollziehbar und überprüfbar, wann von wem auf diese Daten zugegriffen wurde. Dies verbessert die Möglichkeiten der Nutzer im Hinblick auf eigenverantwortlichen Datenschutz.

Sharing Economy

Ziel der Sharing Economy ist es, ungenutzte Vermögenswerte gemeinsam zu nutzen, Produkte und Dienstleistungen bereitzustellen und Angebot und Nachfrage zusammenzuführen, sowohl profitorientiert als auch gemeinnützig. [SYZ16] betrachten dies am Beispiel Smart City und stellen konzeptionell vor, wie entsprechende Geschäftsmodelle auf Basis von BC Technologien umgesetzt werden können.

Sammeln und Verteilen von Daten und Informationen

Das Sammeln und Verteilen von Daten und Informationen im Hinblick z.B. auf Kapazitäten, Verfügbarkeit, Ausnutzungsgrad, Messwerten o.ä. ist in BC Architekturen transparent abbildbar. Es stellt einen eigenständigen Anwendungsfall dar, ist gleichzeitig aber auch Grundlage für die Umsetzung von Sharing Economy Geschäftsmodellen.

4.2. Konkrete Anwendungsszenarien

Einen konkreteren Überblick über Einsatzmöglichkeiten von Blockchains und Smart Contracts im Umfeld von IoT geben [CD16]. So beschreiben sie ein Szenario für den Firmware Update von IoT Geräten unabhängig von einer zentralen Serverstruktur des Herstellers und sie skizzieren verschiedene Geschäftsmodelle auf Basis der direkten Abwicklung von Transaktionen zwischen IoT Geräten.

Bereitstellung von Speicherplatz

Filecoin ist ein dezentrales Speichernetzwerk, das über eine Blockchain mit einem nativen Protokoll-Token (Filecoin), den Handel mit Speicherplatz ermöglicht [PL18].

Die Autoren reklamieren, dass das BC Protokoll durch Speicherreplikation im Netzwerk Datenverlust vermeidet und Replikfehler automatisch behebt. Alle Daten werden verschlüsselt abgespeichert, der Schlüssel ist nur dem Client bekannt.

Bereitstellung von Rechenleistung

21 ist eine Micropayment BC Plattform, die unter Nutzung von Bitcoins den Zugriff auf Application Programming Interfaces (APIs) zur Durchführung von Berechnungen erlaubt [Gra+16].

Handel mit physischen Ressourcen

slock.it ist Anbieter des Universal Sharing Network (USN). Ziel des Unternehmens ist es, jedermann zu ermöglichen, ungenutzte Kapazitäten von physischen Gütern (Fahrzeuge, Wohnungen u.ä.) zu vermieten, ohne dass der Vermieter oder ein Mittelsmann dabei selbst in Erscheinung treten muss [SLOCK.IT]. Die physischen Güter erhalten eine Identität im IoT und über Ethereum Blockchain Transaktionen die Fähigkeit, Zahlungen zu empfangen und zu senden. Die Kommunikation zur Freischaltung der physischen Ressource (z.B. das Öffnen des Schlosses) erfolgt über anonyme Nachrichten, die über die BC ausgetauscht werden [WHISPER].

Management von IoT Geräten

Eine konkrete Umsetzung für das Management von IoT Geräten im Umfeld einer Smart Home Anwendung beschreibt [HCK17]. Die Autoren demonstrieren in einer Simulation, wie über Smart Contracts in der Ethereum BC IoT Geräte Informationen Austauschen und sich selbständig adaptiv konfigurieren.

Informationssicherheit im Smart Home und Smart City Umfeld

Dorri et al. schlagen eine spezifische Blockchain Architektur vor: ein BC Overlay Netzwerk, das die Kommunikation zwischen Smart Home Devices und Cloud Speicher vermittelt [DKJ17; Dor+17]. Sie reklamieren, dass dadurch Vertraulichkeit, Integrität und Verfügbarkeit in Smart Home Anwendungen ohne signifikanten zusätzlichen Aufwand deutlich gesteigert werden können.
Als Sicherheitsframework für Smart City Applikationen schlagen [BM16b] vor, auf den verschiedenen Layern des Technologiestacks unterschiedliche Methoden anzuwenden und insbesondere unterschiedliche Blockchains für die Kommunikation einerseits und die Speicherung von Daten andererseits vorzusehen. Die Kommunikation erfolgt verschlüsselt und anonymisiert, die Speicherung der Daten kann je nach Da-

tenart und Anwendung in einer öffentlichen, einer privaten oder einer Konsortium BC verarbeitet werden.

Supply Chain Tracking

[KL16] stellen in Form eines Proof-of-Concept ein Echtzeit-Tracking System vor, das identifizierbare Ressourceneinheiten (tracable ressource unit (TRU)) zurückverfolgen kann auf alle anderen vorhergehenden TRUs in der Produktionskette. Sie implemetieren dies in Form einer Distributed Application (DAPP)[2] auf der Ethereum BC und basierend auf einem ontologischen Modellierungsansatz.

Industrie 4.0 Anwendungen

[BM16a] stellt ein IoT Device auf Basis eines ein Platinen Computers (z.B. Raspberry Pi) vor, dass als Gateway zwischen heutigen Industrie 4.0 Anwendungen und der Blockchain dient und so verschiedenste Industrie 4.0 Anwendungen um die vorteilhaften Eigenschaften der BC wie verbindliches Logging und Transaktionen erweitert:

- On-Demand Manufacturing,
- Diagnose und Wartung,
- Nachverfolgbarkeit von Produktionsabläufen,
- Lieferkettenmanagement,
- Produktzertifizierung,
- Nutzer-Maschine und Maschine-Maschine Transaktionen,
- Hersteller-Qualitätstracking und
- Inventarisierung und Konfigurationsmanagement.

BC Anwendungen in der Energiewirtschaft

[GRID+17] avisiert ebenfalls ein Device, dass als Gateway zwischen dem Smart Meter und der Ethereum Blockchain agiert. Dieser „smart energy agent“ ermöglicht u.a., Energiebezug in Echtzeit zu bezahlen und damit automatisiert auf kurzer Zeitskala den jeweils günstigsten Anbieter zu wählen. Sie nutzen dabei die Technologie der State Channels. Hierbei wird nicht für jeden einzelnen Zahlvorgang eine eigene Transaktion ausgelöst, sondern über eine Transaktion wird ein State Channel geöffnet, über den

[2] DAPPS sind verteilte Anwendungen, die auf der Blockchain Architektur implementiert werden und in der Regel ein Web basiertes Nutzerinterface zur Verfügung stellen.

dann eine größere Anzahl von Zahlvorgängen in einer Summe abgewickelt wird. Dies reduziert den Aufwand und die Kosten in der Ethereum BC deutlich.
Weitere Anwendungsbeispiele listet [BDEW17]. Sie decken eine große Bandbreite der bereits allgemein skizzierten Nutzungsszenarien im Kontext der Energiewirtschaft ab:

- Ladeinfrastruktur für Elektromobilität,
- Handel mit Zertifikaten für erneuerbare und regionale Stromproduktion,
- Aufbau und Betrieb von Nachbarschaftsstromnetzen,
- Stromhandel von privat zu privat,
- Einbeziehung von dezentralen Kleinstanlagen in das Engpassmanagement der Energieversorger zur Stabilisierung des Stromnetzbetriebs,
- Überwachung und Dokumentation von Anlagenzuständen,
- Predictive Maintenance,
- manipulationssichere und verteilte Speicherung von Eigentumsverhältnissen,
- Verwaltung von Zugriffs- und Nutzungsrechten,
- Erkennen individueller Verhaltensmuster,
- Bereitstellung von Daten zur Weiterverarbeitung oder zum Verkauf,
- Selbstverwaltung von IoT Geräten,
- Hybridlösungen, in denen die Zugriffsrechte auf Datenbanken in der BC verwaltet werden und
- Demand Side Management (Ladesäulenkoordination zur Verhinderung von Stromnachfragespitzen).

5. Simulation ausgewählter Anwendungsszenarien

Um die Chancen und möglichen Probleme in der Realisation einer Anwendung besser bewerten zu können, werden zwei Szenarien simuliert. Zunächst wird im folgenden kurz die Entwicklungs- und Simulationsumgebung dokumentiert, bevor dann die Simulationen im Detail vorgestellt werden.

5.1. Entwicklungs- und Simulationsumgebung

5.1.1. Private Simulations-Blockchain

Zur Simulation wird eine private Ethereum-Blockchain betrieben. Dazu wird unter Nutzung des Programmes `geth` in der Version 1.8.1-stable-1e67410e [GoEthereum] auf einem Rechner unter Windows 10 ein lokaler BC-Knoten initialisiert:

```
geth --datadir=./chaindata init genesis.json
```

Die Initialisierungsdatei `genesis.json` ist in der Anlage als Listing A.1 beigefügt.
`geth` ist eine freie Implementierung des Ethereum Protokolls in der Programmiersprache Go. Die Quellen sind verfügbar unter [gethSource], Dokumentation im Wiki [gethWiki].
Einmalig initialisiert kann der Knoten mittels

```
geth --datadir=./chaindata --rpc --rpcaddr "0.0.0.0"
       --rpcport "8545" --rpccorsdomain "*" --gcmode archive
       --rpcapi "db,eth,net,personal,miner,web3"
```

gestartet werden. Die Kommandozeilenparameter tragen dafür Sorge, dass dieser Knoten über das Netzwerk per Remote Procedure Call (RPC) angesprochen werden kann. Die Simulationen werden im lokalen Heimnetz durchgeführt. Der BC-Knoten wird innerhalb dieses Netzes unter der Internet Protokoll (IP)-Adresse

```
192.168.1.107:8545
```

adressiert. So kann beispielsweise mittels

```
geth attach http://192.168.1.107:8545
> miner.start(1)
> miner.stop()
```

eine geth Konsole gestartet und von dieser Konsole aus das Mining des Knotens gestartet und gestoppt werden.

5.1.2. Smart Contract Entwicklung

Die Smart Contracts werden in der Hochsprache Solidity [Sol18] entwickelt. Als Entwicklungs- und Testumgebungen stehen Online Remix [Remix] und ETHFiddle [ETHFiddle] zur Verfügung. In diesen Umgebungen kann Solidity Code compiliert werden und es ist (mit Einschränkungen) möglich, die Funktionen des Smart Contract zu testen. Hierzu stellen diese IDEs eine eigene Simulations-BC zur Verfügung. Generell erweist sich Test und Debugging eines Smart Contracts als aufwendig und schwierig. Syntaxfehler werden zur Compilezeit identifiziert, aber Funktionen, wie man sie von einem „normalen“ Debugger kennt (Break- und Watchpoints o.ä.), darf man nicht erwarten. Auch die Nutzung von Textausgabefunktionen zum Logging von Debug-Informationen ist nicht möglich.

5.1.3. Administration der Blockchain Accounts und Verträge

Die Administration der Blockchain, z.B. das Anlegen von Accounts, die Abfrage von Accountguthaben u.ä. kann über die geth Konsole erfolgen. Komfortabler ist dies über den Mist Browser [MistDownload; MistSource; MistWiki]. Der Mist Browser erlaubt über eine graphische Oberfläche u.a.

- das Anlegen von Benutzeraccounts,
- die Abfrage von Accountguthaben,
- die Initiierung von Transaktionen zur Übertragung von Ether,
- das Deployment von Smart Contracts,
- die Initiierung von Transaktionen an Verträge und
- die Abfrage von Events, die über Verträge generiert werden.

5.1.4. Simulation des IoT Endgerätes

Das IoT Endgerät wird über eine virtuelle Maschine (VM) mit Ubuntu 16.04 abgebildet, die im „Bridged Network Modus“ in einer Virtual Box Umgebung auf dem BC

Rechner läuft. Die Kommunikation zwischen der VM und dem Host Rechner ist so nur über das lokale Netzwerk möglich und die VM kann als unabhängiger separater Rechner im Netz betrachtet werden.

5.1.5. Anbindung von simulierten IoT Endgeräten

Für die Anbindung eines IoT Endgerätes an das Blockchain Netzwerk gibt es drei grundsätzliche Optionen, die in Bild 5.1 illustriert sind.
Prinzipiell kann das IoT Gerät selbst einen vollständigen so genannten Full Node (FN) implementieren, der als eigenständiger Knoten im BC Netzwerk agiert und alle Aufgaben eines solchen Knotens (inkl. Mining) wahrnehmen kann. Dies ist allerdings nicht sinnvoll, da IoT Geräte üblicherweise über beschränkte Ressourcen verfügen und ein BC Knoten signifikanten Speicherplatz benötigt. [EthereumStats] listet eine Datenbankgröße der Ethereum BC von 639 GB am Tag des Abrufs der Webseite.
Eine mögliche Alternative ist die Installation eines so genannten Light Node (LN) auf dem IoT Gerät. Ein LN Protokoll für die Ethereum BC ist aktuell in der Entwicklung [ETHLightClient]. Ein solcher LN implementiert ebenso wie ein FN einen BC Knoten. Er lädt regelmäßig alle Block Header, jedoch im Gegensatz zum FN keine Transaktionsdaten. Nur, wenn spezielle Transaktionsdaten für den LN von Interesse sind, fragt er diese zur Verifikation von einem FN im BC Netzwerk an. Wenn ein stabiler ausgetesteter LN zur Verfügung steht, ist dies sicher in verschiedenen Anwendungsszenarien die bevorzugte Wahl zur Einbindung eines IoT Gerätes.
Für die Simulation wurde eine anderer Weg beschritten, der auch in realen Szenarien eine Option darstellt. Die simulierten IoT Geräte konnektieren über RPCs auf einen FN oder einen LN. Dieser dient den IoT Geräten als Gateway ins BC Netzwerk. In einem Smart Home Szenario könnte dieser FN/LN zum Beispiel auf der Smart Home Zentrale implementiert sein.
Eine prinzipiell ähnliche Lösung stellt [Jen18] vor. Hier dienen verschiedene spezielle Knoten in der BC als öffentliche IoT Server, auf die sich IoT Geräte direkt per RPC verbinden können. Das vorgeschlagene Verfahren sieht dabei verschiedenen Maßnahmen vor, die das Vertrauen der IoT Geräte in diese öffentlichen Knoten sicherstellen sollen.

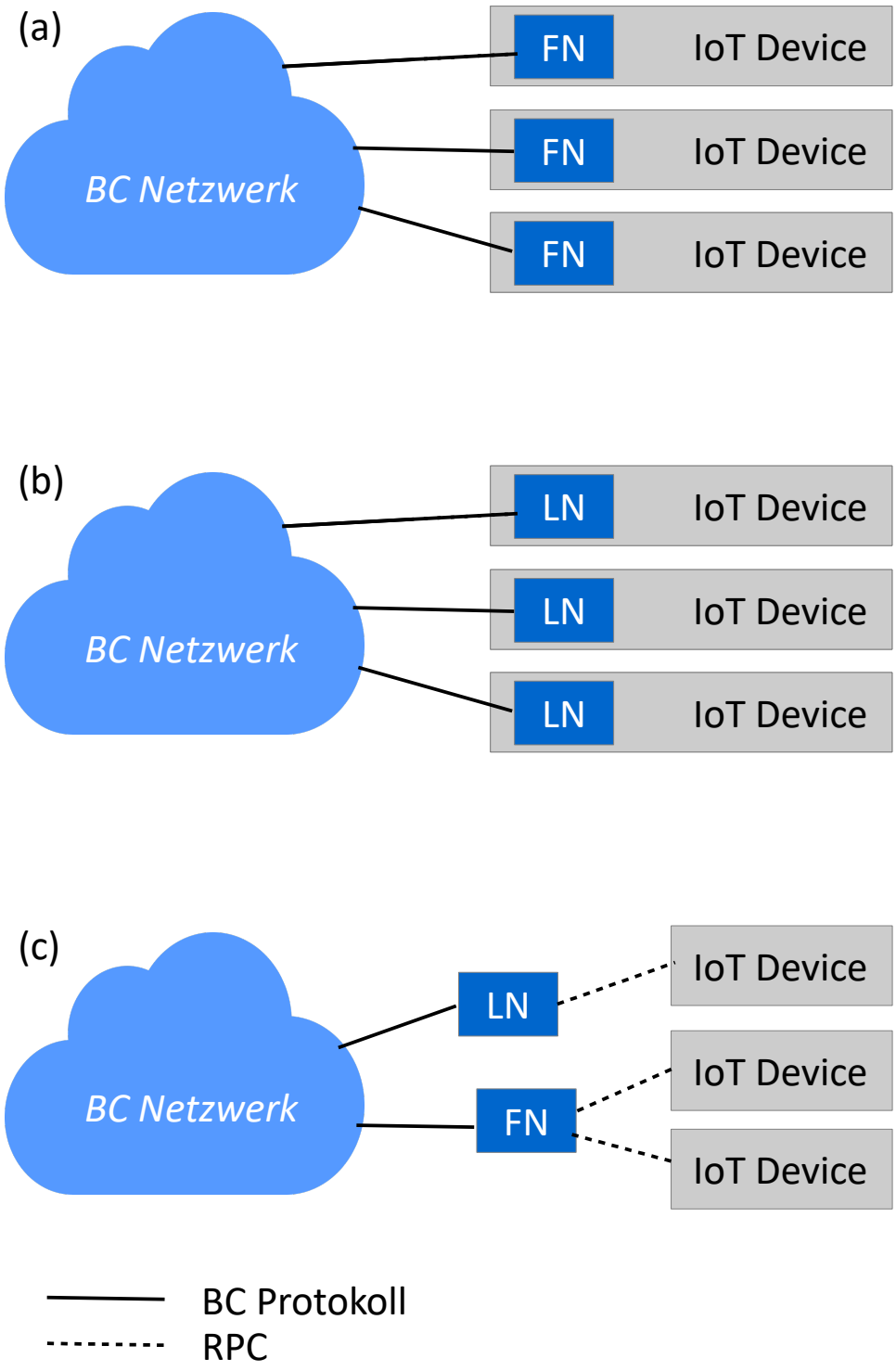


Bild 5.1.: Anbindungsoptionen von IoT Geräten an das Blockchain Netzwerk: (a) via Full Node auf dem IoT Gerät (b) via Light Node auf dem IoT Gerät oder (c) via RPC auf einen Full Node oder Light Node als Gateway.

5.1.6. Web3 ABI

Die Kommunikation eines IoT Endgerätes mit der Blockchain wird über Python Skripte realisiert, die das Web3 Application Binary Interface (ABI) [Web3.py; Web3Doc] nutzen. Diese Python Bibliothek stellt Funktionen zur Verfügung, um u.a.

- per RPC auf einen BC Knoten zu verbinden,
- Transaktionen zu initiieren und so Vertragsfunktionen zu aktivieren und
- von Verträgen generierte Events abzufragen.

5.1.7. IPFS

Die Blockchain Architektur ist darauf ausgelegt, Transaktionsdaten zu speichern. Größere Einzeldatensätze können jedoch nicht über die BC gespeichert und verteilt werden.
Das InterPlanetary File System (IPFS) ist eine Technologie, die genau darauf ausgerichtet ist [Ben14]. IPFS ist ein verteiltes Peer-to-Peer Dateisystem, in dem Dateien über ihren Hash Wert adressiert werden können. Es ist insofern komplementär zur BC, fußt aber auf den gleichen Grundsätzen der Dezentralität und ergänzt funktional die BC in idealer Weise.
Für die Simulationsumgebung wurde auf dem BC Rechner ein IPFS Knoten und auf dem IoT Gerät ein IPFS get Client zum Abruf von Dateien aus dem IPFS installiert [IPFSDownload; IPFSGet; IPFSDocs].

5.2. Szenario 1: Firmware Updates für IoT Geräten

Das erste betrachtete Szenario implementiert die Option, auf IoT Geräten Firmware Updates zu installieren, ohne dazu auf die Verfügbarkeit eines Download-Servers des Herstellers angewiesen zu sein.

5.2.1. Szenariosetup

- In einem Smart Contract wird die Versionsnummer und der Hash der jeweils aktuellen Firmware hinterlegt.
- IoT Geräte können diesen Hashwert über eine Transaktion an den Vertrag abfragen.
- Weicht der abgefragte Hash vom Hash der installierten FW Version ab, lädt das IoT Endgerät die aktuelle Firmware Version über das IPFS.

Verwendet werden

- die private Ethereum Test BC mit einem Smart Contract zur Realisierung der skizzierten Funktionalität,
- die VM als IoT Gerät, angebunden an die BC via RPC, mit einem Python Skript zur Simulation der Endgerätefunktionalität,
- der Mist Browser zur Verwaltung der BC und
- IPFS zur Bereitstellung der Firmware Dateien.

5.2.2. Smart Contract

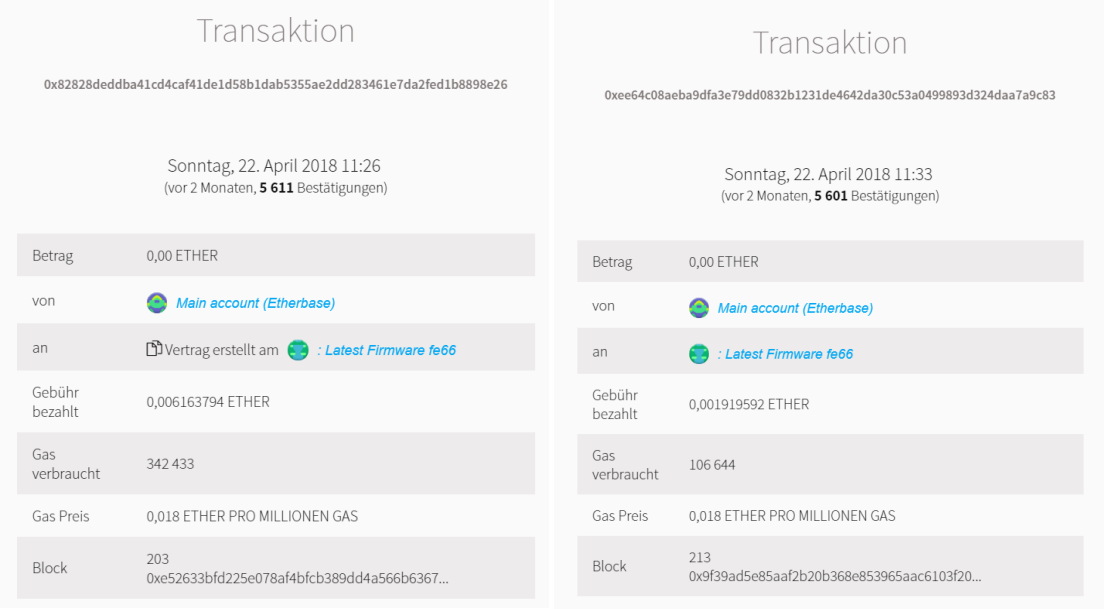
Der Smart Contract kann einfach gehalten werden, siehe Listing 5.1. Lediglich der Hash der Firmwaredatei sowie die Firmwareversion werden als Zustand gespeichert. Diese Werte können nur von demjenigen aktualisiert werden, der den Vertrag angelegt hat, abfragen kann sie jedermann.

Listing 5.1: Smart Contract `LatestFirmware.sol`

```
pragma solidity ^0.4.0;

contract LatestFirmware {
  address public publisher;
  string public FirmwareHash;
  uint16 public FirmwareVersion;

  // constructor; registriere publisher des Contracts
  function LatestFirmware() public {
      publisher = msg.sender;
  }

  function set_LatestFirmware(string fwh, uint16 v) public {
      // nur der publisher kann die Daten ändern
      require(msg.sender == publisher);
      FirmwareHash = fwh;
      FirmwareVersion = v;
  }

  // nur der publisher kann den Contract löschen
  function killme() public {
      require(msg.sender == publisher);
      selfdestruct(msg.sender);
  }
}
```

Das ABI des Vertrages wird z.B. über Remix (zur Compilezeit) oder von Mist (zur Deployzeit) zur Verfügung gestellt. Es ist für den Smart Contract nach Listing 5.1 in Anhang B.1 gelistet.

Via Mist Browser wird der Vertrag in der BC deployed und per weiterer Transaktion die Firmware Version 1 sowie der Hash der zugehörigen Firmwaredatei als aktuelle Version im Vertrag hinterlegt. Die entsprechenden Transaktionen sind in den Screenshots in Bild 5.2 dokumentiert.



(a) Vertragsanlage (b) Vertragstransaktion

Bild 5.2.: Im Mist Browser angezeigte Transaktionsinformationen zur Anlage des Vertrages und zum Update der Versionsinformationen.

5.2.3. IoT Simulation

Als IoT Endgerät wird die VM genutzt. Die aktuelle Firmware wird über eine Datei `firmware.txt` simuliert, die auf diesem IoT Gerät liegt (Listing 5.2). Der Firmware Status nach Version und Hash ist in einer weiteren Datei `current_fw.json` abgelegt (Listing 5.3).

Listing 5.2: Simulierte Firmware-Datei `firmware.txt`

```
Dies ist die Firmware Version 1.
```

Die Endgerätefunktionalität wird über das Python Script nach Listing 5.4 realisiert. Als Input wird benötigt

- die Adresse des RPC Interfaces des Gateway Knotens,

Listing 5.3: Firmware Status auf dem IoT Gerät in `current_fw.json`

```
{"hash": "QmRqqSLdFuMnoALVAi9YvcEEMQmXzudxb1oW1VFfvUzMur", "
   version": 1}
```

- die Adresse des Vertrages in der BC und
- das ABI des Vertrages.

Das Skript

1. prüft die aktuelle lokale Firmware-Version (Lesen der Datei `current_fw.json`) und
2. fragt per Transaktion an den Smart Contract die dort hinterlegte Firmware-Version und den Firmware-Hash ab. Liegt eine aktuellere Version vor,
3. lädt das Skript per `ipget` diese Version aus dem IPFS,
4. überschreibt die alte Firmware-Datei (als Simulation des Update-Prozesses) und
5. aktualisiert den lokalen Status in der Datei `current_fw.json`.

IPFS

Über den IPFS Knoten auf dem BC Rechner werden drei Firmware-Dateien (Version 1, 2 und 3) in das IPFS geladen:

```
>ipfs add firmware1.txt
34 B / 34 B [============] 100.00% 0s
added QmRqqSLdFuMnoALVAi9YvcEEMQmXzudxb1oW1VFfvUzMur
firmware1.txt

>ipfs add firmware2.txt
34 B / ? [-------------]
added QmSLe9ecmZojjCSMYkwwXXZ7GxZzwKpwEVNRM4HxgZMBJ5
firmware2.txt

>ipfs add firmware3.txt
34 B / 34 B [============] 100.00% 0s
added Qmb5FRkQTJ82YRf9s9zkvMQcgjzYZtxr7wFEAuFgExPKeR
firmware3.txt
```

Listing 5.4: Python Script `fw.py` zur Simulation des IoT Endgerätes

```
# Python3 Skript auf IoT Gerät
from web3 import Web3, HTTPProvider
from hexbytes import HexBytes
from subprocess import run
import json

RPCinterface = 'http://192.168.1.107:8545'
contract_address = '0xfE66eb2F510b4d89B5c408495Ad5a2d58a75A3A4
   '

# aktuelle fw hash und version laden
with open('current_fw.json', 'r') as fw_state:
  current_fw = json.load(fw_state)

print("Aktuelle␣FW␣Version:␣{}".format(current_fw["version"]))

# Verbindung zur BC aufbauen
w3 = Web3(HTTPProvider(RPCinterface))

# contract api initialisieren
with open('latestfw_abi.json', 'r') as abi_definition:
  contract_abi = json.load(abi_definition)

contract = w3.eth.contract(address = contract_address, abi =
   contract_abi)

# contract state variablen abrufen
fwhash = contract.functions.FirmwareHash().call()
fwversion = contract.functions.FirmwareVersion().call()

print("Im␣Contract␣hinterlegte␣FW␣Version:␣{}".format(
   fwversion))

if fwversion > current_fw["version"]:
  print("lade␣neue␣FW␣Version")
  filepath = '/ipfs/'+fwhash
  if run(['./ipget', filepath, '-o', 'fwdownload.txt']).
     returncode == 0:
    current_fw["hash"] = fwhash
    current_fw["version"] = fwversion
    with open('current_fw.json', 'w') as fw_state:
      json.dump(current_fw, fw_state)
    run(['mv', '-b', 'fwdownload.txt', 'firmware.txt'])
else:
  print("Firmware␣up␣to␣date,␣Version:␣{}".format(fwversion))

# Ausgabe der aktuellen Datei firmware.txt
print('firmware.txt:')
run(['cat', 'firmware.txt'])
```

5.2.4. Simulationslauf

Die Ausgangssituation der Simulation ist:

- Die Firmware Version auf dem IoT Gerät ist 1.
- Die im Vertrag hinterlegte Version ist 1.

Ein Start des Skriptes `fw.py` liefert den erwarteten Output, es erfolgt kein Update:

```
dn@dn-VirtualBox:~$ python3 fw.py
Aktuelle FW Version: 1
Im Contract hinterlegte FW Version: 1
Firmware up to date, Version: 1
firmware.txt:
Dies ist die Firmware Version 1.
```

Nach Aktualisierung der FW Version via Mist im Vertrag auf Version 2 erfolgt ein Update:

```
dn@dn-VirtualBox:~$ python3 fw.py
Aktuelle FW Version: 1
Im Contract hinterlegte FW Version: 2
lade neue FW Version
firmware.txt:
Dies ist die Firmware Version 2.
```

Ein erneuter Lauf von `fw.py` in diesem Status liefert erwartungsgemäß

```
dn@dn-VirtualBox:~$ python3 fw.py
Aktuelle FW Version: 2
Im Contract hinterlegte FW Version: 2
Firmware up to date, Version: 2
firmware.txt:
Dies ist die Firmware Version 2.
```

und nach Aktualisierung im Vertrag auf Version 3 erfolgt auch auf dem Endgerät ein neuerlicher Update:

```
dn@dn-VirtualBox:~$ python3 fw.py
Aktuelle FW Version: 2
Im Contract hinterlegte FW Version: 3
lade neue FW Version
firmware.txt:
Dies ist die Firmware Version 3.
```

5.2.5. Feststellungen

In Verallgemeinerung dieses Anwendungsszenarios können folgende Punkte festgehalten werden:

- Wenn Information (wie hier die Firmware-Version eines IoT Gerätes) öffentlich zugänglich gemacht werden soll, kann eine öffentliche Blockchain benutzt werden.
- Die Anlage eines Smart Contracts zur Verwaltung und Bereitstellung solcher Informationen in der BC erfordert keine großen Aufwände.
- Der Abruf der Informationen erfordert kein Account auf der BC. Lediglich das Account des Vertrages und das ABI des Vertrages muss bekannt sein und eine Anbindung an das BC Netzwerk bestehen.
- Für den reinen Abruf der Informationen fallen (in der Ethereum BC) keine Kosten an.
- Das Deployen des Smart Contracts sowie nachfolgende Updates der im Vertrag hinterlegten Informationen via an den Vertrag adressierte Transaktionen erfordern einen Account auf der BC.
- Diese Aktionen sind mit Kosten (in Ether) verbunden. Der Account muss über ein entsprechendes Guthaben verfügen. Nach Bild 5.2 belaufen sich die Kosten für das Deployment des Vertrages auf ca. 0,0062 Ether und für einen Update der Informationen auf ca. 0,0019 Ether. [EthereumStats] listet einen Kurs von ca. 621 US$ / Ether am Tag des Abrufs der Webseite. Damit ergeben sich Transaktionskosten von ca. 3,85 US$ respektive 1,18 US$.
- Die Information sind dezentral in der BC hinterlegt und damit unabhängig von der Infrastruktur des Herstellers verfügbar.
- Die Nutzung von IPFS erlaubt, auch große Datenmengen dezentral und unabhängig von eigener Infrastruktur bereitzustellen. Dies ergänzt die Möglichkeiten der BC.
- Zur Implementierung der Kommunikation des IoT Endgerätes mit einem BC Knoten oder dem Abruf von Dateien aus dem IPFS stehen Standardlibraries und Open Source Software zur Verfügung. Die Nutzung dieser Funktionen gestaltet sich unaufwändig.

5.3. Szenario 2: Aggregation von Sensordaten

Mit der zunehmende Verbreitung von Smart City und Smart Home Anwendungen werden auch immer mehr Sensoren installiert, die Informationen über die Umwelt aufnehmen. Im folgenden wird ein Szenario betrachtet, um diese Informationen für die Allgemeinheit oder für Dritte nutzbar zu machen.
Die Sensoren senden ihre Sensordaten bei Überschreitung eines Grenzwertes an einen Smart Contract. Der Smart Contract bündelt diese Informationen z.B. nach regionalen Kriterien und generiert seinerseits einen Event, wenn in einer Region in einem bestimmten Zeitraum eine kritische Anzahl an Meldungen überschritten wird. Dritte können diese Events verfolgen. Je dichter das Netz an Sensoren, die sich beteiligen, um so zuverlässiger die Warnhinweise.

5.3.1. Szenariosetup

Für die Simulation werden folgende Annahmen getroffen:

- Es werden insgesamt 20 Sensoren in 4 Regionen simuliert. Jeweils fünf Sensoren werden einer Region zugeordnet.
- Ein Sensor kann sich als Melder beim Smart Contract für seine Region registrieren.
- Im Smart Contract ist das Level für kritische Messwerte hinterlegt. Es kann vom Vertrags-Eigner geändert werden.
- Das Level kann von den Sensoren abgefragt werden. Ein Sensor meldet, wenn er einen Messwert registriert, der diese Warnschwelle überschreitet.
- Der Smart Contract akzeptiert je Account nur eine Meldung innerhalb von 24 Stunden und setzt Sensoren, die häufiger senden, auf eine Blacklist. Dies dient der Vermeidung von Spam.
- Der Smart Contract generiert einen Warn-Event, wenn innerhalb der Zeit von 5 Stunden mehr als 40 Prozent der für eine Region registrierten Sensoren (in der Simulation also 2 Sensoren in einer Region) eine Überschreitung melden. Weitere Warnevents für diese Region werden für die nächsten 5 Stunden unterdrückt.
- Sendet ein Sensor eine Transaktion, die den Zustand der BC verändert (Registrierung oder Messwert), ist diese Transaktion in der Ethereum BC mit Kosten verbunden. Die Sensoraccounts müssen daher mit einem initialen Guthaben ausgestattet sein. Um dieses Guthaben in der Balance zu halten, wird dem Sen-

soraccount nach einer erfolgreichen Transaktion durch den Vertrag eine Rückvergütung in einem Mindestumfang gewährt, die die Kosten für eine nächste Transaktion abdeckt.

- In der Simulation entspricht ein Tag 24 Minuten, 1 Stunde also 1 Minute.
- Jede Stunde (+/-) wird von jedem Sensor ein Messwert generiert. Für jede Region gibt es eine ausgewählte Stunde, in der der Messwert mit einer Wahrscheinlichkeit von 80% über dem kritischen Wert liegt. Es ist also wahrscheinlich, aber nicht sicher, dass jeden Tag für jede Region zu einer bestimmten Stunde die kritische Anzahl an Meldungen für eine Warnung überschritten wird.
- Die Messwerte liegen die restlichen 23 Stunden am Tag mit einer Wahrscheinlichkeit von 80% unter dem kritischen Wert. Auch außerhalb der „kritischen“ Stunde kann es daher dazu kommen, dass zwei Sensoren innerhalb von 5 Stunden eine Meldung absetzen und eine Warnung ausgegeben wird.
- Der Sensor sendet kritische Werte an den Smart Contract, wenn seine letzte Meldung mindestens 23 Stunden zurückliegt. Es kann also zu Blacklisting kommen.
- Es wird der Verlauf über zwei Tage simuliert.

Verwendet werden die private Ethereum Test BC mit einem Smart Contract zur Realisierung der skizzierten Funktionalität und die VM für die Simulation aller Sensoren. Dies stellt keine Einschränkung dar, da jeder Sensor in der VM in einem eigenen Thread realisiert wird und die Absender IP-Adresse im Prozess keine Rolle spielt. Die Sensorfunktionalität wird wieder über Python Skripte abgebildet und die Sensoren konnektieren wie in Szenario 1 via RPC auf die BC.

5.3.2. Smart Contract

Der Smart Contract `Sensorwarner7.sol` ist in Listing C.1 in der Anlage angeführt. Zur Realisierung der gewünschten Funktionalität implementiert der Vertrag unter anderem die Datenstrukturen

```
// Datenstruktur je Sensor
struct SensorAccount {
  uint region;                    // Region des sensors
  bool isRegistered;
  bool isBlacklisted;
  uint contributions;             // Anzahl gesendeter Messungen
  uint lastMeasureTime;           // Zeitpunkt letzte Meldung
}
```

```
// Verwaltung der registrierte Sensoren
mapping (address => SensorAccount) public sensors;
// Datenstruktur je Messung
struct Measure {
  uint measurement;              // Messwert
  uint time;                     // Zeitpunkt (des Blocks)
}
// Array zur Speicherung der Messungen je Region
Measure[][4] measurements;
```

und die Funktionen

```
// Setter zur Aenderung des Levels für kritische Messwerte
function set_level(uint _level) public {}
// Setter zur Aenderung der Rueckverguetung
function set_refund(uint _refund) public {}
// Registrierungsanfrage
function register(uint _region) public {}
// Erhaltene Meldung
function putMeasure(uint measure) public {}
```

Die Funktionen `set_level()` und `set_refund()` gestatten es dem Vertragseigner, das Warnlevel und die Höhe der Rückvergütung zu setzen. Die Funktion `register()` dient der Registrierung eines Sensors. Die wesentliche Vertragslogik ist in der Funktion `putMeasure()` abgebildet. Ein Sensor sendet einen Messwert, indem er per Transaktion diese Funktion anspricht. Zunächst wird geprüft, ob der Sensor registriert ist und nicht auf der Blacklist steht. Ferner wird geprüft, ob der Messwert über dem kritischen Level liegt. Ist eine dieser Bedingungen nicht erfüllt, wird die Transaktion abgewiesen. Wenn die letzte Meldung des Sensors weniger als 24 Stunden zurückliegt, wird er auf die Blacklist gesetzt und seine Meldung ebenfalls verworfen. Ansonsten wird die Meldung mit ihrem Zeitstempel aufgenommen und die Transaktion zur Rückvergütung angestoßen. In Folge wird ein Warn-Event erzeugt, wenn innerhalb der letzten 5 Stunden eine hinreichende Anzahl an Meldungen für eine Warnung eingegangen und bislang in dieser Zeit bislang auch noch keine Warnung erfolgt ist.
Neben dem Warn-Event werden auch noch Events definiert, die nach einer erfolgreichen Registrierung und nach einer erfolgreichen Meldung abgesetzt werden:

```
event MeldungErhalten();
event RegistrierungErhalten();
event Warning();
```

Das ABI des Vertrages ist als Listing C.2 der Anlage beigefügt.

5.3.3. IoT Simulation

Zur Simulation der Endgerätefunktionalität wurden verschiedene Python Skripte implementiert. Alle Skripte verbinden jeweils per RPC auf den Test-BC Knoten und benötigen dazu als Input die Adresse des RPC Interfaces:

```
RPCinterface = 'http://192.168.1.107:8545'
```

Die nachfolgend aufgeführten und kurz hinsichtlich ihrer Funktionalität erläuterten Skripte sind in der Anlage C gelistet.

gen_accounts.py

Dieses Skript legt auf dem Test-BC Knoten 20 neue Accounts an und speichert die Account Adressen in der Datei `sensor_accounts.json`.

initial_ether.py

Dieses Skript stattet die Sensoraccounts mit einem initialen Guthaben aus, indem von einem Account mit genügendem Guthaben („coinbase“) entsprechende Beträge tranferiert werden. Um dies tun zu können, benötigt das Skript das Passwort des coinbase Accounts, das per RPC übertragen wird.

register_sensors.py

Dieses Skript registriert alle Sensoren im Vertrag. Es weißt dabei die Sensoren jeweils einer Region zu. Benötigt wird die Account-Adresse des Sensors, die Adresse des Vertrages sowie das Vertrags-ABI:

```
contract_address = '0x2C273a3c9802244D16dcDaBDF1309aD28458a777'
abi_file = 'sensorwarner_abi.json'
```

Ferner muss zur Signierung der Transaktion das Account Passwort per RPC übertragen werden.

sim_sensor.py

Dieses Skript simuliert einen einzelnen Sensor über den Zeitverlauf von zwei Tagen (entsprechend 48 Minuten in der Simulation) mit dem in Abschnitt 5.3.1 beschriebenen zufälligen Mess- und Meldeverhalten. Auch dieses Skript benötigt Kenntnis über die Account Adresse des Sensors, die Adresse des Vertrages sowie das Vertrags-ABI. Alle Transaktionen zur Meldung einer Messung müssen signiert werden. Dazu muss das Account-PW per RPC übertragen werden. Die Messungen bzw. Meldungen werden in einem csv-Format ausgegeben.

parallel.py

Dieses Skript startet in 20 parallelen Prozessen jeweils das Skript `sim_sensor.py`. Durch das zufallsgesteuerte Zeitverhalten der einzelnen Sensoren ergibt sich ein unabhängiges asynchrones Verhalten. Die Ausgaben der einzelnen Sensoren werden jeweils in eine Datei umgelenkt.

eventlog.py

Dieses Skript liest aus der BC alle vom Vertrag generierten Events und loggt sie in der Datei `event.log`. Als Input wird lediglich die Vertrags-Adresse und das Vertrags-ABI benötigt. Für die Abfrage von Events ist kein Account erforderlich, es fallen keine Kosten an.

get_sensor_status.py

Dieses Skript liest den im Vertrag gespeicherten Status aller Sensoren aus und schreibt ihn nach `stdout`.

5.3.4. Simulationslauf

Der Simulationsablauf gestaltet sich wie folgt:

1. Deployen des Smart Contracts.
2. Generierung der Accounts mittels `gen_accounts.py`. Das Ergebnis in Form der Datei `sensor_accounts.json` ist als Listing C.10 in der Anlage beigefügt.
3. Ausstattung der Accounts mit einem initialen Guthaben via Skript `initial_ether.py`.
4. Alle Sensoren im Vertrag registrieren mittels Skript `register_sensors.py`.
5. Starten der Simulation via Skript `parallel.py`.

5.3.5. Auswertung der Simulation

Nach Abschluss des Simulationslaufs wurden zunächst die Ausgaben der einzelnen Sensoren in einer gemeinsamen Datei `sensors.log` zusammengeführt. Tabelle 5.1 zeigt einen exemplarischen Ausschnitt in Tabellenform.

Tabelle 5.1.: Ausschnitt des Sensorlogs in tabellarischer Form mit den Werten h:min:sec (Absoluter Zeitstempel Realzeit), m (simulierte Messstunde), Wert (Messwert zwischen 0 und 100), Account (Account-Adresse in abgekürzter Form), Region (0-3) und gesendet (Messwert an Vertrag gesendet True/False)

h	min	sec	m	Wert	Account	Region	gesendet
7	8	38	9	71	0x0fB2...	1	False
7	25	6	16	61	0xa2A2...	2	False
7	30	41	21	71	0x5a27...	3	True

Die vollständige Datei ist der Arbeit als digitale Anlage beigefügt.
Mittels `get_sensor_status.py` wurde der Status der Sensoren nach der Simulation abgerufen. Die vollständige Ausgabe des Skripts ist als Listing C.11 der Anlage beigefügt. Nachfolgend ist exemplarisch der Auszug für zwei Sensoren gelistet.

```
Sensorstatus for sensor12
@ 0x27bf892f566AAE03E5Ff328FB0a26E4D51C178f6:
----------------------------------------------
Region: 0
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2

Sensorstatus for sensor17
@ 0x0fB265216640286115707bF089089Db93CC4948a:
----------------------------------------------
Region: 1
isRegistered: True
isBlacklisted: True
Anzahl Meldungen: 1
```

Zwei der 20 Sensoren wurden im Verlauf der Simulation auf die Blacklist gesetzt.
Das Eventlog wurde mittels `eventlog.py` abgerufen. Das vollständige Eventlog ist der Arbeit als digitale Anlage beigefügt, einen Ausschnitt zeigt Tabelle 5.2.

Tabelle 5.2.: Beispielhafte Auszüge aus dem Eventlog für jeweils einen Eventtyp Registrierung, Meldung, Warnung

Event	Beispiel
Registrierung	Mon May 14 06:56:52 2018, Block: 5394, Event: RegistrierungErhalten, Details: AttributeDict('Region': 1, 'from': '0x64F5947fBAb008689BD93acd9709E0396657656a')
Meldung	Mon May 14 07:01:16 2018, Block: 5431, Event: MeldungErhalten, Details: AttributeDict('Meldung': 70, 'Region': 3, 'numToCheck': 2, 'numTotal': 1, 'from': '0xFaA3A4021059B1E025d8419f25EB92C21D72b946', 'Meldungsintervall': 0)
Warnung	Mon May 14 07:02:23 2018, Block: 5439, Event: Warning, Details: AttributeDict('region': 1, 'numOfMeasures': 2, 'wlevel': 3)

Das Eventlog wurde manuell mit dem Sensorlog abgeglichen. Das Ergebnis ist in der Datei `sensorlog2.xlsx` dokumentiert und entspricht dem erwarteten Verhalten. Auch das Blacklisting der beiden Sensoren ist auf Basis der Logs nachvollziehbar. Die Auswertedatei ist der Arbeit als digitale Anlage beigefügt. Einen exemplarischen Auszug zeigt Bild 5.3. Beim Vergleich der Zeitstempel in den Logs ist zu berücksichtigen, dass alle Zeitstempel in der BC sich immer auf den Block beziehen, in dem die entsprechende Transaktion protokolliert ist.

h	m	s	h sim	Mess.	Sensor Adresse	Region	Gesendet	Kommentar	Warning-Event aus Eventlog	Meldungs-Event aus Eventlog
6	59	12	14	71	0xBCda938C17586501D5e59fC57c4FEE83B0B05623	1	True	erste Meldung R 1 -> keine Warnung		Mon May 14 06:59:20 2018, Block: 5414, Event: MeldungErhalten, Details: AttributeDict({'Meldung': 71, 'Region': 1, 'numToCheck': 2, 'numTotal': 1, 'from': '0xBCda938C17586501D5e59fC57c4FEE83B0B05623', 'Meldungsintervall': 0})
6	59	13	14	71	0xA4311Ec9A405a7345Cc310b07401B3e9c82040ce	0	True	erste Meldung R 0 -> keine Warnung		Mon May 14 06:59:20 2018, Block: 5414, Event: MeldungErhalten, Details: AttributeDict({'Meldung': 71, 'Region': 0, 'numToCheck': 2, 'numTotal': 1, 'from': '0xA4311Ec9A405a7345Cc310b07401B3e9c82040ce', 'Meldungsintervall': 0})
7	0	12	15	70	0x87D0a11D8968B6D065cDe3C239d79eDa97DD8981	2	True	erste Meldung R 2 -> keine Warnung		Mon May 14 07:00:12 2018, Block: 5424, Event: MeldungErhalten, Details: AttributeDict({'Meldung': 70, 'Region': 2, 'numToCheck': 2, 'numTotal': 1, 'from': '0x87D0a11D8968B6D065cDe3C239d79eDa97DD8981', 'Meldungsintervall': 0})
7	0	16	15	70	0xf5AA4E7F3a59494	2	True	zweite Meldung R 2 < 5 Min -> Warnung erwartet	Mon May 14 07:00:20 2018, Block: 5425, Event: Warning, Details: AttributeDict({'region': 2,	Mon May 14 07:00:20 2018, Block: 5425, Event: MeldungErhalten, Details: AttributeDict({'Meldung': 70,
7	0	20	15	71	0xfbB942F30023CBc09b6b8E4a3778028c49ff78fB	0	True	zweite Meldung R 0 in 5 Min -> Warnung erwartet	Mon May 14 07:00:20 2018, Block: 5425, Event: Warning, Details: AttributeDict({'region': 0, 'numOfMeasures': 2, 'wlevel': 3})	Mon May 14 07:00:20 2018, Block: 5425, Event: MeldungErhalten, Details: AttributeDict({'Meldung': 71, 'Region': 0, 'numToCheck': 2, 'numTotal': 2, 'from': '0xfbB942F30023CBc09b6b8E4a3778028c49ff78fB', 'Meldungsintervall': 60})
7	1	5	16	70	0xFaA3A4021059B1E025d8419f25EB92C21D72b946	3	True	erste Meldung R 3 -> keine Warnung		Mon May 14 07:01:16 2018, Block: 5431, Event: MeldungErhalten, Details: AttributeDict({'Meldung': 70, 'Region': 3, 'numToCheck': 2, 'numTotal': 1, 'from': '0xFaA3A4021059B1E025d8419f25EB92C21D72b946', 'Meldungsintervall': 0})

Bild 5.3.: Exemplarischer Auszug des Abgleichs von Sensorlog und Eventlog.

5.3.6. Feststellungen

In Verallgemeinerung der Simulationsergebnisse kann festgehalten werden:

- Grundsätzlich ist die öffentliche Ethereum BC geeignet, um solche Szenarien abzubilden.
- Ein Problem hierbei können die Transaktionskosten sein. Die Transaktionsgebühr für eine Meldung beläuft sich auf ca. 0,0022 Ether entsprechend ca. 1,367 US$ [EthereumStats]. Das ist in einem realen Szenario wirtschaftlich nicht abbildbar. Eine Alternative ist die Nutzung einer Konsortium-BC, die z.B. von verschiedenen Smart Home Anbietern, Verbraucher- und Umweltorganisationen, der öffentlichen Hand, Versicherungen oder anderen gemeinsam betrieben wird. Auch hier kann eine virtuelle digitale Währung genutzt werden, um Spamming zu vermeiden und Besitzer von Sensoren durch Incentivierung an der Teilnahme am Netzwerk zu motivieren. Die Anbieter der Konsortium-BC können dann aber selbst und frei über den „Wert“ dieser digitalen Währung entscheiden.
- Ein weiterer Nachteil bei Nutzung der öffentlich BC ist, das jeder Teilnehmer zur Registrierung bereits über Guthaben verfügen muss. Dies ist in einer Konsortium BC auch anders darstellbar. Es wird diskutiert, auch in Ethereum zukünftig die Funktionalität „Vertrag zahlt die Transaktionskosten“ zu realisieren [But15].
- Spam kann verhindert werden, indem im Smart Contract je Account nur n Meldungen in der Zeit t zugelassen werden. Kombiniert mit dem Kostenmodell ergibt sich ein effektiver Spam-Schutz, da für den Absender in jedem Fall Kosten entstehen, auch wenn seine Transaktion nicht verarbeitet wird.
- Im Smart Contract kann eine Blacklist oder eine Whitelist zugelassener Accounts verwaltet werden.
- Alle Transaktionen in einem Block tragen einen gemeinsamen Zeitstempel: den des Blocks. Eine feinere Auflösung kann nicht erreicht werden. Der Zeitstempel des Blocks wird immer von dem Knoten vergeben, der den Block signiert. Da Knoten nicht zwingend zeitsynchron laufen, können sich hier zusätzliche Zeitdifferenzen ergeben. Jede Auswertung von Zeitstempeln muss diese Unsicherheit berücksichtigen.
- Die Realisierung des Szenarios im vorgestellten Setup erfordert die Übertragung der Account-Passwörter über RPC. Unter Sicherheitsaspekten ist dies kritisch zu bewerten. Wird ein anderes Anbindungsszenario realisiert, z.B. ein Light Node direkt auf dem Endgerät, muss „nur“ gewährleistet werden, dass die Accountdaten im Gerät nicht abgreifbar sind.

- Die Abfrage und Verfolgung von Events ist nicht an einen Account gebunden und für jedermann kostenfrei möglich.

6. Ergebnisse und Diskussion

Die Wertschöpfung des IoT ergibt sich aus der Verarbeitung von Transaktionen zwischen den vernetzten Dingen und Nutzern. Blockchain Technologien sind hierfür ideal geeignet. Sie bieten in diesem Kontext zwei wesentliche Mehrwerte:

1. Sie ermöglichen neue Geschäftmodelle, indem sie automatisierte regelbasierte Transaktionen ohne Eingriff eines Mittelsmannes ermöglichen und dabei Zahlfunktionen unterstützen.

2. Sie steigern die Informationssicherheit, indem sie die Transaktionen validieren und unveränderbar, geordnet und jederzeit überprüfbar protokollieren. Sie sind dabei resistent gegenüber byzantinem Verhalten einzelner Akteure.

So verwundert es nicht, dass der Einsatz von Blockchain Technologien für das IoT Gegenstand aktueller wissenschaftlicher Forschungen ist und sich auch schon dokumentierte Anwendungen finden, die produktiv sind oder sich in der Testphase vor einer breiten Einführung befinden.
Die Ansätze zeigen auch: Es gibt nicht die eine Technologie sondern variable Ausprägungen – öffentlich, privat oder konsortial, unterschiedliche Consensus Verfahren – mit jeweiligen Vor- und Nachteilen in Abhängigkeit der Anwendung.
Öffentliche Blockchains wie Ethereum können als wesentlichen Vorteil verbuchen: Sie sind verfügbar und können von jedermann genutzt werden. Es ist – unterstützt durch Open Source Tools und Dokumentation – ohne große Aufwände möglich, einfache Anwendungen zu implementieren, wie die in Abschnitt 5 vorgestellten Simulationen zeigen. Und so einfach das erste dieser Beispiele – die Verwaltung und Verteilung von Firmware Updates – auch gestaltet sein mag: Schon heute könnten Hersteller für sich damit Kostenvorteile erzielen, indem keine eigene Infrastruktur zur Softwareverteilung vorgehalten werden muss. Und für die Nutzer tritt ein automatischer Sicherheitsgewinn dadurch ein, dass IoT Geräte auch dann noch ihre Software aktualisieren können, wenn der Besuch der Update-Webseite des Herstellers auf eine 404 Fehler führt[1]. Dabei sind die Aufwände und Kosten für diesen Anwendungsfall minimal, und auf Seiten des

[1] Über die 404 Fehlermeldung wird im http Protokoll angezeigt, dass eine angefragte Ressource nicht verfügbar ist.

Endgerätes ist nicht einmal eine Account auf der Ethereum Blockchain von Nöten, da nur Statusinformationen abgefragt werden.
Das zweite Simulationsbeispiel – die Auswertung von Sensordaten – zeigt aber auch direkt die Einschränkungen bei der Nutzung der öffentlichen Ethereum Blockchain, sobald das Szenario etwas komplizierter wird. Die technische Umsetzung selbst ist auch hier im wesentlichen unkritisch. Wenn allerdings die Kosten für eine Transaktion, die den Status der Blockchain verändern, je nach Ether Kurs mit mehr als einem US$ zu Buche schlagen, ist dies nicht wirtschaftlich darstellbar. Hier müssen zwangsläufig private oder konsortiale BC eingesetzt werden.
Anbieter, die öffentliche BCs nutzen, umgehen dieses wirtschaftliche Problem teilweise, in dem sie State Channels auf der Ethereum Blockchain nutzen und z.B. 1000 Bezahlvorgänge in einer Transaktion bündeln [GRID+17]. Dies löst aber nicht das grundsätzliche Problem. Auch erscheint es nicht sinnvoll, spezifische Probleme durch „spezielle Features“ wie State Channels zu lösen. Vielmehr ist es sinnvoll, die grundsätzlichen Konzepte der Blockchain Technologien weiter zu entwickeln, um aktuelle Einschränkungen zu lösen.
Als weiteres positives Ergebnis der Simulationen ist festzuhalten: die Möglichkeit, (missbräuchliche) BC Transaktionen mit Kosten zu belegen, kann ein effektiver Spam-Schutz sein.

6.1. Technologisch bedingte Einschränkungen

6.1.1. Vertraulichkeit

Eine Stärke der Blockchain, die Transparenz, ist gleichzeitig eine potentielle Schwäche. Accounts sind pseudonymisiert, aber die Nutzung bezogen auf das Pseudonym (den öffentlichen Schlüssel eines Accounts) ist allen Nutzern der Blockchain verfügbar. Eine Analyse von Verhaltensmustern kann Angreifern wertvolle Rückschlüsse auf die Identität eines Accounts oder zumindest über seine Aktivitäten erlauben. Dieses Problem wurde bereits von Nakamoto in seinem Bitcoin Paper [Nak08] adressiert. Sein Vorschlag hierzu ist, als Nutzer für jede Transaktion einen neuen Account zu verwenden. Dies mag für den reinen Handel mit Bitcoins eine Option sein, da hier der öffentliche und private Schlüssel lediglich zur Authentisierung, nicht jedoch zur Authorisierung verwendet werden. Im IoT Kontext ist Authorisierung jedoch zwingend, und damit ist ein ständiger Wechsel der Identität durch einen neuen Account nicht möglich.

6.1.2. Skalierbarkeit

Per Definition kann das Datenvolumen einer Blockchain immer nur wachsen, und die Rechenleistung der gesamten Blockchain ist unabhängig von der Anzahl der Knoten immer nur so groß wie die Rechenleistung eines einzelnen Knotens, da ja jeder Knoten (FN) alle Blockchain Operationen berechnet[2]. Hinzufügen weiterer Knoten erhöht die Ausfallsicherheit und verringert die Möglichkeit byzantiner Angriffe, trägt aber nicht zur Steigerung der Rechenleistung bei. Dies limitiert zwangsläufig die Fähigkeit zu skalieren. Als Option, Skalierbarkeit zu ermöglichen, wird Sharding diskutiert: das Teilen großer BC in Sub-BCs, die in einer übergeordneten BC zusammengeführt werden. Für Ethereum existieren konkrete Umsetzungspläne, allerdings stellt dieses Vorgehen einen gewissen Bruch zur ursprünglichen Idee eines verteilten Netzes dar und Auswirkungen auf die BC Eigenschaften müssen detailliert betrachtet werden [Sharding].

6.1.3. Validierungszeit

In Zusammenhang mit der Skalierbarkeit steht die Validierungszeit: Wie lange dauert es, bis ein Block validiert wird und wie lange dauert es, bis eine mögliche Race Condition aufgelöst wird. Die Zeit, um einen Block zu validieren, wird beim Proof-of-Work Consensus durch den Schwierigkeitsparameter gesetzt. Er ist ein Regulativum, um steigende Rechenkraft durch technologischen Fortschritt auszugleichen und so das Vertrauen in der BC sicherzustellen. In der Bitcoin BC wird der Schwierigkeitsparameter so eingestellt, dass die erwartete Zeit für die Validierung eines Blockes ca. 10 Minuten beträgt. Grund dafür ist, die Wahrscheinlichkeit von Race Conditions zu verringern unter Berücksichtigung der Ausbreitungszeit von Informationen über das gesamte BC Netzwerk. Basierend auf Erfahrungswerten aus der Vergangenheit ist eine potentielle Race Condition mit hoher Wahrscheinlichkeit nach sechs Blöcken aufgelöst. In der Bitcoin BC kann daher eine Transaktion erst nach ca. einer Stunde als final validiert betrachtet werden. In der Ethereum BC liegt die Blockzeit bei ca. 10 - 20 Sekunden. Dies funktioniert, da hier verbesserte Verfahren zum Umgang mit der Race Condition implementiert wurden [But14; SZ15].
Die Validierungszeit ist beim Proof-of-Work Consensus ein Designkriterium, das wesentlich für die Sicherheit der Blockchain ist. Der Proof-of-Work Mechanismus funktioniert, weil eine Mindestzeit für die Validierung eines Blockes benötigt wird. Je nach Anwendungsfall kann die Länge der Validierungszeit ein Ausschlusskriterium für eine bestimmte Blockchain-Variante sein. Andere Consensus Verfahren sind in dieser

[2] Dies gilt natürlich nicht für die Mining Rechenleistung. Diese bestimmt sich über die Summe der Rechenleistung aller Full Nodes.

Hinsicht unkritischer.

6.1.4. De-Dezentralisierung

Das Beispiel Bitcoin BC zeigt, dass der Grad der Dezentralisierung, der mit wachsender BC eigentlich immer weiter steigen sollte, eher gegenläufig ist. Mining wird nur noch von wenigen großen Anbietern betrieben, weil es sich für kleinere Anbieter nicht mehr rechnet. Das Prinzip des Vertrauens durch Consensus in einem hochgradig verteilten System wird ad absurdum geführt. Wesentliche Ursache ist das Consensus Verfahren Proof-of-Work. Es besteht die Hoffnung, dass andere Consensus Protokolle in dieser Hinsicht besseres Verhalten zeigen.

6.2. IT Sicherheitsrisiken der Blockchain

Blockchains sind aufgrund ihrer vielfältigen vorteilhaften Eigenschaften im Hinblick auf die Verbesserung der Informationssicherheit hilfreich. Es ist jedoch wichtig, sich vor Augen zu halten, dass IoT Infrastruktur gegenüber klassischen Angriffsszenarien zunächst auch weiterhin durch klassisches IT Security Management abgesichert werden muss und Blockchains Hersteller und Diensteanbieter hier nicht aus dieser Verpflichtung entbinden. Der Einsatz von BCs hilft nicht in diesem Bereich. Ihr Vorteil liegt vielmehr darin, Datennutzung kontrollierbar, transparent und überprüfbar zu machen. Auf der anderen Seite ist natürlich die BC selbst auch Angriffen ausgesetzt und es ergeben sich zusätzliche BC spezifische Risiken.

6.2.1. Schlüsselmanagement

Außer in einfachsten Anwendungsfällen, in denen ein IoT Gerät nur auf Informationen in der BC zugreift, selbst aber keine statusverändernden Transaktionen initiiert, benötigt das IoT Gerät einen Account, der im wesentlichen aus einem öffentlichen und privaten Schlüsselpaar besteht. Hat ein Angreifer Zugriff auf den privaten Schlüssel, kann er im Namen des Accounts des IoT Gerätes beliebige manipulierte Transaktionen in der BC durchführen. Im Hinblick auf die IT-Sicherheit ist dies ein extrem kritischer Faktor, da das Schlüsselmanagement für IoT Devices wie in Abschnitt 2 erläutert eines der Hauptprobleme im IoT ist. Wie können Schlüssel sicher auf die IoT Geräte verbracht und dort sicher verwahrt werden? Ebenso kritisch ist der Verlust eines privaten Schlüssels. Jeder Zugriff auf Werte im Account ist nur über den privaten Schlüssel möglich. Ebenso wird in vielen Lösungen der private Schlüssel zur Authorisierung von Funktionen innerhalb eines Smart Contracts dienen. Verliert z.B. Alice aus dem Smart Contract Beispiel aus Abschnitt 3.5 ihren privaten Schlüssel, kann sie weder auf

die Zertifkate in ihrem Account zugreifen noch die Funktion `verkaufeDochNicht()` aktivieren.
Hilfreich in diesem Zusammenhang - auch unabhängig von der Verwaltung der Schlüssel für die BC - können Multiparty Secure Computing Algorithmen sein, bei dem der private Schlüssel verteilt abgespeichert wird und die Signatur einer Transaktion erfolgt, ohne dass die verteilten Schlüsselteile zusammengeführt werden müssen. Ein Angreifer, der Zugriff auf das IoT Gerät erhält, kann so nur maximal über einen für sich genommen wertlosen Teilschlüssel verfügen. Die Grundzüge dieser Verfahren wurden bereits 1982 von Yao beschrieben [Yao82], heute sind effiziente kommerzielle Lösungen verfügbar [YLxx].

6.2.2. Korrektheit von Smart Contracts

Der Code eines einmal in der BC eingestellten Smart Contracts kann nachträglich nicht geändert werden. Er kann auch nicht mehr gelöscht werden, außer im Code selbst ist eine entsprechende Funktion zum Löschen des Vertrages vorgesehen. Dies ist eine wichtige Eigenschaft von Smart Contracts, denn alle Parteien, die den Smart Contract zur Abwicklung gemeinsamer Transaktionen nutzen, müssen darauf vertrauen können, dass die vereinbarten und im Smart Contract niedergelegten Regelungen unverändert sind. Ein Problem stellt dies jedoch dann dar, wenn der Smart Contract einen Programmierfehler enthält, entweder weil

1. eine Funktionalität falsch implementiert wurde oder
2. eine relevante Regelung im Vertrag „vergessen“ wurde.

Würde der in Abschnitt 3.5 betrachtete Beispiel-Smart Contract zum Verkauf von Zertifikat A von Alice an Bob beispielsweise keine Regelung `verkaufeDochNicht()` vorsehen, hätte Alice nach der Übertragung Ihres Zertifikates auf den Vertrag keine Möglichkeit mehr, Ihre Zertifikat wieder rückzuübertragen, sollte Bob seiner Zahlungsverpflichtung nicht nachkommen. Das Zertifikat würde für immer und ewig dem Vertrag gehören.
Berühmtes Beispiel für den ersten Fall - ein Bug in einem Smart Contract - ist die DAO Attacke [But16b; Mad18]. Die fehlerhafte Implementierung einer Funktion in einem komplexen Smart Contract erlaubte es einem Angreifer, 3,6 Millionen Ether im damaligen Gegenwert von ca. 70 Millionen US$ auf ein anderes Konto zu übertragen. Die Situation konnte letztlich nur dadurch gelöst werden, dass ein so genannter Hard Fork der Ethereum BC durchgeführt wurde: Ethereum Knoten spielten eine neue (nicht abwärtskompatible) Ethereum Software Version auf, die auf der Blockchain vor dem Hack aufsetzte. Da sich nicht alle Knoten an diesem Software Update beteiligten, kam

es zu einem Split der Ethereum BC. Neben der Ethereum BC gibt es seither auch die Ethereum Classic BC.
Beide Beispiele zeigen, das es extrem wichtig ist, einen Smart Contract intensiv zu prüfen, bevor er deployed wird. Einen formalen Ansatz hierzu beschreibt [CGV18]. Weitere Empfehlungen zur Entwicklung sicherer Smart Contracts sowie hilfreicher Tools zur Verifikation beschreibt [SCBestPractice].

6.2.3. Fehler in der Implementierung des Blockchain Protokolls

Wie jede Software kann und wird auch die BC Implementierung Fehler enthalten. Es ist immer nur eine Frage der Zeit, bis diese Fehler entdeckt und potentiell missbräuchlich ausgenutzt werden. Wird ein solcher Fehler erkannt und im Code korrigiert, muss zudem jeder Knoten auf die korrigierte Software Version updaten. Ein weiterer Aspekt ist, dass es nicht „die eine“ Implementierung gibt. Für Ethereum gibt es beispielsweise Versionen in Go, Rust, C++, Python, Javascript, Java und Ruby, und es steht jedem frei, seine eigene Version des BC Protokolls zu implementieren.

6.2.4. Bekannte Angriffsvektoren

Bekannte Angriffsvektoren sind unter anderem

51% Attacken

Gelingt es jemandem, die Mehrheit im Consensus-Verfahren zu übernehmen, hat er damit Effektiv die Hoheit darüber, welche Transaktionen validiert werden. Das Risiko ist insbesondere für kleinere BC Netzwerke real. So wurden im Mai 2018 erfolgreich mehrere 51% Attacken gegen die Bitcoin Gold BC geführt [TN18].

Double Spending Attacken

Mallory plaziert die gleiche Transaktion (z.B. die Übertragung von Kryptowährung) gleichzeitig einmal an Alice, von der er dafür eine Gegenleistung erwartet (T_1), und zum anderen an sich selbst (T_2). Wenn diese beiden Transaktionen während einer Race Condition in zwei parallelen Ketten verarbeitet werden, sind sie jeweils für sich genommen scheinbar validiert. Für Knoten, die Kette 1 als gültig erachten, ist T_1 gültig, Für Knoten, die Kette 2 als gültig erachten, T_2. Nach Auflösung der Race Condition wird eine der beiden Transaktionen verworfen. Wenn sich T_2 durchsetzt, Alice jedoch im Vertrauen auf die Validierung in Kette 1 vorher schon ihre Leistung erbracht hat, erhält sie nun keine Bezahlung dafür. Dies ist der Grund, warum für die Bitcoin BC

empfohlen wird, erst nach 6 Folgeblöcken eine Transaktion als final validiert zu betrachten. Besonders kritisch ist dieser Angriff in Kombination mit einer 51% Attacke. Beim schon erwähnten Angriff geben das Bitcoin Gold Netzwerk nutzen die Angreifer ihre Mehrheit an der Rechenleistung, um eine verdeckte Race Condition auszulösen: Sie bauten eine zweite parallele BC_P auf, ohne diese zunächst im Netzwerk zu veröffentlichen. In der zu diesem Zeitpunkt „netzwerkoffiziellen“ BC_O platzierten sie Transaktionen T_1 zum Verkauf von Bitcoin Gold gegen US$. In ihrer eigenen parallelen BC_P platzierten sie die Übertragung T_2 der identischen Bitcoin Gold an sich selbst. Nachdem der Tausch gemäß T_1 ausgeführt worden war, veröffentlichten die Angreifer ihre BC_P, die aufgrund ihrer Rechenleistung inzwischen länger war als BC_O. Alle Transaktionen der BC_O seit Beginn der erzwungenen Race Condition verloren damit ihre Gültigkeit. Der Schaden für die Händler, die die Bitcoin Gold gegen reale US$ getauscht hatten, belief sich auf ca. 18 Millionen US$ [TN18].

Selfish Mining Attacken

Bei der Selfish Mining Attacke handelt es sich um eine Strategie, bei der eine Gruppe von Minern validierte Blöcke zunächst zurückhält und nicht mit allen Knoten teilt. Sie erzwingen damit also ebenfalls eine Race Condition. Bei der Selfish Mining Attacke nutzen die Angreifer dies jedoch nicht als Basis für eine Manipulation der Transaktionen, sondern um für sich unter Umständen Vorteile hinischtlich der Mining Incentives zu erzielen [ES13].

Routing Attacken

Routing Attacken zielen darauf ab, die Netzwerkverbindungen von Knoten zu sabotieren oder den Austausch der Nachrichten zwischen den Knoten zu verzögern [AZV17].

Eclipse Attacken

Bei dieser Form von Attacke werden die eingehenden und ausgehenden Verbindungen eines Knotens „gekapert“ [Hei+15]. Der Knoten kann mit manipulierten Nachrichten belegt werden.

6.3. Übergreifende Aspekte

In Abschnitt 2 wurden als besondere Herausforderungen im Hinblick auf die IT-Sicherheit des IoT drei Aspekte identifiziert:

- die Sicherstellung der Kontrolle über die vernetzten Dinge,

- die Wahrung der Hoheit über die Daten und
- der Schutz von Schlüsselmaterial und Zugangsdaten.

Blockchains können nicht alle dieser Probleme lösen. Ein verantwortungsvolles klassisches IT-Sicherheitsmanagement ist weiterhin eine zwingende Basis. Neben der Option, als Enabler für neue Geschäftsmodelle zu dienen, können sie aber einen wichtigen Beitrag im Hinblick auf eigenverantwortlichen Datenschutz bieten und die Abhängigkeit von einzelnen Plattformanbietern reduzieren.
Die Möglichkeiten des Einsatzes von Blockchain Technologien in jedem IoT Device sind heute noch limitiert durch Ressourcenbeschränkungen auf den IoT Geräten. Unter Verweis auf Moore's Law reklamieren Brody und Pureswaren [BP14], dass diese Einschränkung zukünftig entfällt. Während IoT Devices heute applikationsspezifisch unter Nutzung von Embedded Systemen arbeiten, erwarten Sie einen Wechsel von IoT Geräten hin zu „general purpose computing". Diese Argumentation ist nachvollziehbar. Die heute angebotenen IoT-BC Gateway Devices sind ein Zwischenschritt in diese Richtung. Mit dieser Entwicklung kann der Einsatz von Blockchain Technologie nicht nur für isolierte IoT Anwendungen, sondern als grundlegende Infrastruktur zum Betrieb des gesamten IoT dienen und einen eigenen generischen Layer im Technologiestack des IoT bilden wie in Bild 6.1 illustriert. In diesem Szenario kann auch die Eigenschaft der BC positiv auswirken, byzantine Teilnehmer zu identifizieren und ggf. zu isolieren.

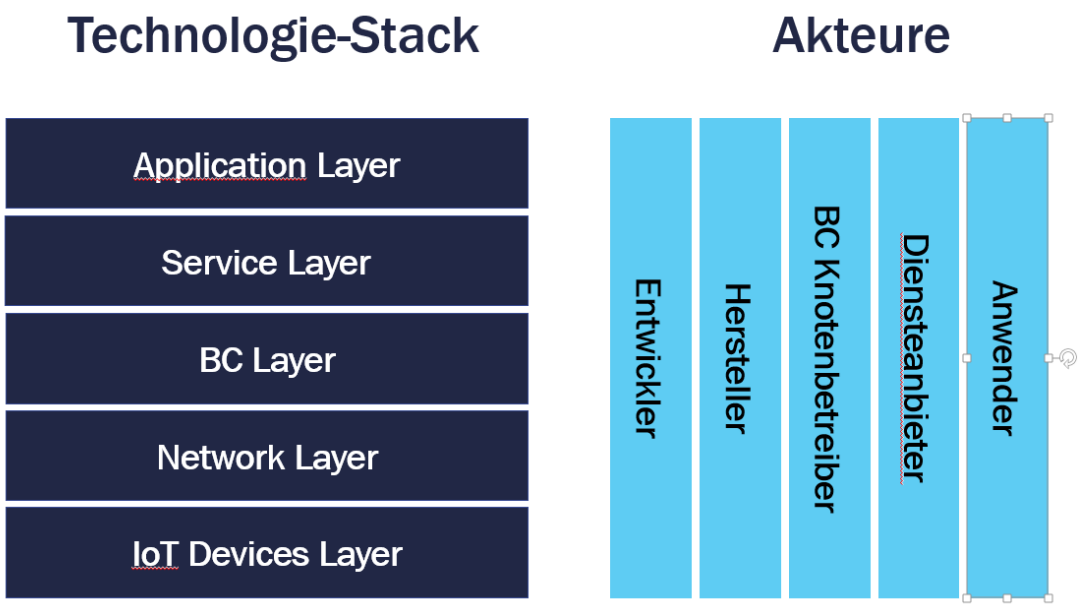


Bild 6.1.: IoT Technologiestack mit Blockchain Layer und beteiligte Akteure

In diesem Zusammenhang ergeben sich dann Fragestellungen zur Interoperabilität von Blockchains. Anwendungen, die heute auf unterschiedlichen Blockchains realisiert

werden, sind weitestgehend Insellösungen. Perspektivisch ist anzustreben, dass diese Lösungen auf Basis eines gemeinsamen Protokolls interagieren können.

7. Zusammenfassung

Das Internet der Dinge verbindet die physische mit der digitalen Welt und eröffnet damit zahlreiche Optionen für fortschrittliche Dienste. Gleichzeitig steigen die Risiken für die Informationssicherheit und den Datenschutz deutlich. Blockchain Technologien erlauben, Daten und Transaktionsregeln in Form von Smart Contracts unveränderlich und kontrollierbar abzulegen. Die Blockchain Architektur ist aufgrund ihrer hochgradig verteilten Peer-to-Peer Infrastruktur hochverfügbar und mittels Consensus Verfahren fehlertolerant gegen einzelne missbräuchlich manipulative Teilnehmer. Blockchains erlauben die automatisierte Abwicklung von Transaktionen wirtschaftlicher und technischer Natur zwischen Teilnehmern, die sich gegenseitig nicht vertrauen müssen und auch ohne dass es der Einschaltung eines vertrauenswürdigen Mittelsmannes bedarf. Die möglichen Anwendungsfälle für Blockchains im IoT Umfeld sind zahlreich, konkrete Umsetzungen heute allerdings noch nicht breit verfügbar. Die Simulation zweier Beispielszenarien zeigt, dass sich die technische Realisierung grundsätzlich unaufwendig gestaltet, allerdings im Detail die konkrete Ausprägung der Blockchain deutlichen Einfluss auf die Umsetzbarkeit eines Anwendungsszenarios hat. Auch, wenn heute noch technische Einschränkungen und auch Risiken bestehen: Als zusätzlicher Layer im IoT Stack wirken Blockchains perspektivisch nicht nur als Geschäftsmodellenabler, sondern stärken auch Informationssicherheit, Datenschutz und Unabhängigkeit von spezifischen Dienstleistern.

Literatur

[And+18] Elli Androulaki u. a. „Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains“. In: *Proceedings of the Thirteenth EuroSys Conference*. EuroSys ’18. Porto, Portugal: ACM, 2018, 30:1–30:15. ISBN: 978-1-4503-5584-1. URL: http://doi.acm.org/10.1145/3190508.3190538 (besucht am 10. 06. 2018).

[Atz+18] Nicola Atzei u. a. „SoK: Unraveling Bitcoin Smart Contracts“. In: *Principles of Security and Trust*. Hrsg. von Lujo Bauer und Ralf Küsters. Cham: Springer International Publishing, 2018, S. 217–242. ISBN: 978-3-319-89722-6.

[AZV17] Maria Apostolaki, Aviv Zohar und Laurent Vanbever. „Hijacking Bitcoin: Routing Attacks on Cryptocurrencies“. In: *2017 IEEE Symposium on Security and Privacy (SP)*. Mai 2017, S. 375–392.

[BDEW17] *Blockchain in der Energiewirtschaft. Potenziale für Energieversorger*. Studie. BDEW Bundesverband der Energie- und Wasserwirtschaft e.V., Okt. 2017. URL: https://www.bdew.de/service/publikationen/blockchain-energiewirtschaft/ (besucht am 16. 06. 2018).

[Ben+14a] Iddo Bentov u. a. *Proof of Activity: Extending Bitcoin’s Proof of Work via Proof of Stake*. Techn. Ber. 2014. URL: https://eprint.iacr.org/2014/452 (besucht am 31. 05. 2018).

[Ben+14b] Iddo Bentov u. a. „Proof of Activity: Extending Bitcoin’s Proof of Work via Proof of Stake [Extended Abstract]“. In: *SIGMETRICS Perform. Eval. Rev.* 42.3 (Dez. 2014), S. 34–37. ISSN: 0163-5999. URL: http://doi.acm.org/10.1145/2695533.2695545.

[Ben14] Juan Benet. „IPFS - Content Addressed, Versioned, P2P File System“. In: *CoRR* abs/1407.3561 (2014). URL: http://arxiv.org/abs/1407.3561 (besucht am 03. 06. 2018).

[BM16a] Arshdeep Bahga und Vijay K. Madisetti. „Blockchain Platform for Industrial Internet of Things“. In: *Journal of Software Engineering and Applications* 9.10 (Okt. 2016), S. 533–546. URL: http://www.scirp.org/journal/PaperDownload.aspx?paperID=71596 (besucht am 17. 06. 2018).

[BM16b] K. Biswas und V. Muthukkumarasamy. „Securing Smart Cities Using Blockchain Technology“. In: *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. Dez. 2016, S. 1392–1393.

[BP14] Paul Brody und Veena Pureswaran. *Device democracy. Saving the future of the Internet of Things*. Techn. Ber. IBM Institute for Business Value, Sep. 2014. URL: https://public.dhe.ibm.com/common/ssi/ecm/gb/en/gbe03620usen/global-business-services-global-business-services-gb-executive-brief-gbe03620usen-20171002.pdf (besucht am 16. 06. 2018).

[But+13] Vitalik Buterin u. a. *A Next-Generation Smart Contract and Decentralized Application Platform*. White Paper. Nov. 2013. URL: https://github.com/ethereum/wiki/wiki/White-Paper (besucht am 24. 05. 2018).

[But14] Vitalik Buterin. *Toward a 12-second Block Time*. Blogpost. 11. Juni 2014. URL: https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time/ (besucht am 20. 06. 2018).

[But15] Vitalik Buterin. *Understanding Serenity, Part I: Abstraction*. Blogpost. 24. Dez. 2015. URL: https://blog.ethereum.org/2015/12/24/understanding-serenity-part-i-abstraction/ (besucht am 12. 06. 2018).

[But16a] Vitalik Buterin. *A Proof of Stake Design Philosophy*. Blogpost. 31. Dez. 2016. URL: https://medium.com/@VitalikButerin/a-proof-of-stake-design-philosophy-506585978d51 (besucht am 16. 06. 2018).

[But16b] Vitalik Buterin. *CRITICAL UPDATE Re: DAO Vulnerability*. Post. 17. Juni 2016. URL: https://pastebin.com/xW16N7Ye (besucht am 18. 06. 2018).

[CD16] K. Christidis und M. Devetsikiotis. „Blockchains and Smart Contracts for the Internet of Things“. In: *IEEE Access* 4 (2016), S. 2292–2303.

[CGV18] Krishnendu Chatterjee, Amir Kafshdar Goharshady und Yaron Velner. „Quantitative Analysis of Smart Contracts“. In: *Programming Languages and Systems*. Hrsg. von Amal Ahmed. Cham: Springer International Publishing, 2018, S. 739–767. ISBN: 978-3-319-89884-1.

[CL02] Miguel Castro und Barbara Liskov. „Practical Byzantine Fault Tolerance and Proactive Recovery“. In: *ACM Trans. Comput. Syst.* 20.4 (Nov. 2002), S. 398–461. ISSN: 0734-2071. URL: http://doi.acm.org/10.1145/571637.571640 (besucht am 10. 06. 2018).

[CV17] Christian Cachin und Marko Vukolic. „Blockchain Consensus Protocols in the Wild“. In: *CoRR* abs/1707.01873 (Juli 2017). URL: https://arxiv.org/abs/1707.01873 (besucht am 10. 06. 2018).

[DH76] Whitfield Diffie und Martin E. Hellman. „New Directions in Cryptography“. In: *IEEE Trans. Inf. Theor.* 22.6 (Nov. 1976), S. 644–654. URL: http://dx.doi.org/10.1109/TIT.1976.1055638.

[DKJ17] Ali Dorri, Salil S. Kanhere und Raja Jurdak. „Towards an Optimized BlockChain for IoT“. In: *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*. Apr. 2017, S. 173–178.

[DLS88] Cynthia Dwork, Nancy Lynch und Larry Stockmeyer. „Consensus in the Presence of Partial Synchrony“. In: *J. ACM* 35.2 (Apr. 1988), S. 288–323. ISSN: 0004-5411. URL: http://doi.acm.org/10.1145/42282.42283 (besucht am 10. 06. 2018).

[Dor+17] Ali Dorri u. a. „Blockchain for IoT security and privacy: The case study of a smart home“. In: *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. März 2017, S. 618–623.

[Dre17] Daniel Drescher. *Blockchain Basics: A Non-Technical Introduction in 25 Steps*. Frankfurt am Main: Apress, 2017. ISBN: 978-1-4842-2603-2.

[drs00] drscholl. *Napster Messages*. 7. Apr. 2000. URL: http://opennap.sourceforge.net/napster.txt (besucht am 27. 05. 2018).

[Eck17] Claudia Eckert. „Cybersicherheit Beyond 2020!“ In: *50 Jahre Universitäts-Informatik in München*. Hrsg. von Arndt Bode u. a. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, S. 1–10. ISBN: 978-3-662-54712-0. URL: https://doi.org/10.1007/978-3-662-54712-0_1.

[ES13] Ittay Eyal und Emin Gün Sirer. „Majority is not Enough: Bitcoin Mining is Vulnerable“. In: *CoRR* abs/1311.0243 (2013). URL: http://arxiv.org/abs/1311.0243 (besucht am 18. 06. 2018).

[Giu+17] Pasquale Giungato u. a. „Current Trends in Sustainability of Bitcoins and Related Blockchain Technology“. In: *Sustainability* 9.12 (Nov. 2017), S. 2214. ISSN: 2071-1050. URL: http://dx.doi.org/10.3390/su9122214.

[Gra+16] John Granata u. a. *The First Micropayments Marketplace*. Blogpost. 12. März 2016. URL: https://medium.com/@earndotcom/the-first-micropayments-marketplace-38c321127d12 (besucht am 16. 06. 2018).

[GRID+17] *Welcome to the Future of Energy*. Whitepaper v2.0. ConsenSys, 2017. URL: https://gridplus.io/assets/Gridwhitepaper.pdf (besucht am 16. 06. 2018).

[HCK17] S. Huh, S. Cho und S. Kim. „Managing IoT devices using blockchain platform“. In: *2017 19th International Conference on Advanced Communication Technology (ICACT)*. Feb. 2017, S. 464–467.

[Hei+15] Ethan Heilman u. a. „Eclipse Attacks on Bitcoin's Peer-to-Peer Network“. In: *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, 2015, S. 129–144. ISBN: 978-1-931971-232. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman (besucht am 18. 06. 2018).

[Irs16] Mohammad Irshad. „A Systematic Review of Information Security Frameworks in the Internet of Things (IoT)“. In: *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on*

Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS). Dez. 2016, S. 1270–1275.

[ITU12] ITU. *Overview of the Internet of things*. Recommendation ITU-T Y.2060. International Telecommunication Union, Juni 2012. URL: https://www.itu.int/rec/T-REC-Y.2060-201206-I (besucht am 25. 05. 2018).

[Jan16] Christoph Paar und Jan Pelzl. *Kryptographie verständlich*. Berlin, Heidelberg: Springer Vieweg, 2016. ISBN: 978-3-662-49296-3.

[Jen18] Christoph Jentzsch. *Slock.it IoT Layer. Enabling the economy of things*. Blogpost. 8. Juni 2018. URL: https://blog.slock.it/slock-it-iot-layer-f305601df963 (besucht am 16. 06. 2018).

[KL16] Henry M. Kim und Marek Laskowski. „Towards an Ontology-Driven Blockchain Design for Supply Chain Provenance“. In: *CoRR* abs/1610.02922 (2016). URL: http://arxiv.org/abs/1610.02922 (besucht am 17. 06. 2018).

[KN18] Sunny King und Scott Nadal. *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake*. Techn. Ber. Mai 2018.

[Lam01] Leslie Lamport. „Paxos Made Simple“. In: (Dez. 2001), S. 51–58. URL: https://www.microsoft.com/en-us/research/publication/paxos-made-simple/ (besucht am 10. 06. 2018).

[LSM82] Leslie Lamport, Robert Shostak und Marshall. „The Byzantine Generals Problem“. In: *ACM Trans. Program. Lang. Syst.* 4.3 (Juli 1982), S. 382–401. ISSN: 0164-0925. URL: http://doi.acm.org/10.1145/357172.357176.

[Mad18] Antonio Madeira. *The DAO, The Hack, The Soft Fork and The Hard Fork*. Post. 18. Mai 2018. URL: https://www.cryptocompare.com/coins/guides/the-dao-the-hack-the-soft-fork-and-the-hard-fork/ (besucht am 18. 06. 2018).

[Map17] Carsten Maple. „Security and privacy in the internet of things“. In: *Journal of Cyber Policy* 2.2 (Aug. 2017), S. 155–184. URL: https://www.tandfonline.com/doi/full/10.1080/23738871.2017.1366536 (besucht am 26. 05. 2018).

[Mer80] Ralph C. Merkle. „Protocols for Public Key Cryptosystems“. In: *1980 IEEE Symposium on Security and Privacy*. Apr. 1980, S. 122–134.

[Mer82] Ralph C. Merkle. „Method of Providing Digital Signatures“. US Patent US4309569A (Vereinigte Staaten von Amerika). Jan. 1982.

[Mil+16] Mitar Milutinovic u. a. „Proof of Luck: An Efficient Blockchain Consensus Protocol“. In: *Proceedings of the 1st Workshop on System Software for Trusted Execution*. SysTEX ’16. Trento, Italy: ACM, 2016, 2:1–2:6. ISBN: 978-1-4503-4670-2. URL: http://doi.acm.org/10.1145/3007788.3007790.

[MJ14] Andrew Miller und Joseph J. LaViola Jr. *Anonymous Byzantine Consensus from Moderately-Hard Puzzles: A Model for Bitcoin*. Techn. Ber. Apr. 2014. URL: https://nakamotoinstitute.org/static/docs/anonymous-byzantine-consensus.pdf (besucht am 30. 05. 2018).

[Nak08] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. Techn. Ber. bitcoin.org, Nov. 2008. URL: https://bitcoin.org/bitcoin.pdf (besucht am 24. 05. 2018).

[Net17] POA Network. *Proof of Authority: consensus model with Identity at Stake*. Blogpost. 2. Nov. 2017. URL: https://medium.com/poa-network/proof-of-authority-consensus-model-with-identity-at-stake-d5bd15463256 (besucht am 17. 06. 2018).

[P4T14] P4Titan. *Slimcoin - A Peer-to-Peer Crypto-Currency with Proof-of-Burn*. White Paper. Mai 2014. URL: http://www.doc.ic.ac.uk/~ids/realdotdot/crypto_papers_etc_worth_reading/proof_of_burn/slimcoin_whitepaper.pdf (besucht am 30. 05. 2018).

[PL18] *Filecoin: A Decentralized Storage Network*. Whitepaper. Protocol Labs, 2. Jan. 2018. URL: https://filecoin.io/filecoin.pdf (besucht am 16. 06. 2018).

[Sch90] Fred B. Schneider. „Implementing Fault-tolerant Services Using the State Machine Approach: A Tutorial“. In: *ACM Comput. Surv.* 22.4 (Dez. 1990), S. 299–319. ISSN: 0360-0300. URL: http://doi.acm.org/10.1145/98163.98167 (besucht am 10. 06. 2018).

[Sol18] *Solidity Documentation. Release 0.4.25*. Manual. 30. Mai 2018. URL: https://media.readthedocs.org/pdf/solidity/develop/solidity.pdf (besucht am 02. 06. 2018).

[SW04] Ralf Steinmetz und Klaus Wehrle. „Peer-to-Peer-Networking &-Computing“. In: *Informatik-Spektrum* 27.1 (Feb. 2004), S. 51–54. ISSN: 1432-122X. URL: https://doi.org/10.1007/s00287-003-0362-9.

[SYZ16] Jianjun Sun, Jiaqi Yan und Kem Z. K. Zhang. „Blockchain-based sharing services: What blockchain technology can contribute to smart cities“. In: *Financial Innovation* 2.1 (Dez. 2016), S. 26. ISSN: 2199-4730. URL: https://doi.org/10.1186/s40854-016-0040-y (besucht am 17. 06. 2016).

[SZ15] Yonatan Sompolinsky und Aviv Zohar. „Secure High-Rate Transaction Processing in Bitcoin“. In: *Financial Cryptography and Data Security. 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*. Hrsg. von R. Böhme und T. Okamoto. Bd. 8975. Berlin, Heidelberg: Springer-Verlag, 2015, S. 507–527. URL: http://fc15.ifca.ai/preproceedings/paper_30.pdf (besucht am 20. 06. 2018).

[Sza94] Nick Szabo. *Smart Contracts*. 1994. URL: https://web.archive.org/web/20160620185026/http://szabo.best.vwh.net/smart.contracts.html (besucht am 24. 05. 2018).

[Sza97] Nick Szabo. „Formalizing and Securing Relationships on Public Networks“. In: *First Monday* 2.9 (1997). ISSN: 13960466. URL: http://journals.uic.edu/ojs/index.php/fm/article/view/548 (besucht am 02. 06. 2018).

[TN18] *Bitcoin Gold 51% Attacked, $18 Million Stolen Through Double Spends*. Blogpost. Trustnotes, 24. Mai 2018. URL: https://www.trustnodes.com/2018/05/24/bitcoin-gold-51-attacked-18-million-stolen-double-spends (besucht am 24. 06. 2018).

[TS16] Florian Tschorsch und Björn Scheuermann. „Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies“. In: *IEEE Communications Surveys Tutorials* 18.3 (Sep. 2016), S. 2084–2123. ISSN: 1553-877X.

[Woo18] Gavin Wood. *Ethereum: A Secure Decentralise Generalised Transaction Ledger - Byzantium Version 745dad8*. Yellow Paper. Mai 2018. URL: https://ethereum.github.io/yellowpaper/paper.pdf (besucht am 24. 05. 2018).

[Yao82] Andrew C. Yao. „Protocols for secure computations“. In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. Nov. 1982, S. 160–164.

[YLxx] Yehuda Lindell. *A Primer in Secure Multiparty Computation*. Whitepaper. Unbound Tech. URL: https://info.unboundtech.com/unbound-mpc-primer (besucht am 18. 06. 2018).

Online-Quellen

[EthereumStats] *Ethereum Statistiken*. URL: https://bitinfocharts.com/de/ethereum/ (besucht am 03. 06. 2018).

[ETHFiddle] *EthFiddle - Solidity IDE in the Browser. Powered By Loom Network*. URL: https://ethfiddle.com/ (besucht am 03. 06. 2018).

[ETHLightClient] *Light Client Protocol*. URL: https://github.com/ethereum/wiki/wiki/Light-client-protocol (besucht am 03. 06. 2018).

[gethSource] *GoEthereum Sourcecode*. URL: https://github.com/ethereum/go-ethereum (besucht am 03. 06. 2018).

[gethWiki] *GoEthereum Wiki*. URL: https://github.com/ethereum/go-ethereum/wiki (besucht am 03. 06. 2018).

[GoEthereum] *GoEthereum Download*. URL: https://gethstore.blob.core.windows.net/builds/geth-windows-amd64-1.8.1-1e67410e.exe (besucht am 24. 02. 2018).

[Hyperledger] *About Hyperledger*. URL: https://www.hyperledger.org/about (besucht am 10. 06. 2018).

[IPFSDocs] *IPFS Docs*. URL: https://ipfs.io/docs/ (besucht am 03. 06. 2018).

[IPFSDownload] *IPFS Download (Win64)*. URL: https://dist.ipfs.io/go-ipfs/v0.4.15/go-ipfs_v0.4.15_windows-amd64.zip (besucht am 03. 06. 2018).

[IPFSGet] *IPFS get Download (Linux)*. URL: https://dist.ipfs.io/ipget/v0.3.0/ipget_v0.3.0_linux-amd64.tar.gz (besucht am 03. 06. 2018).

[MistDownload] *Mist Download Link (Win64)*. URL: https://github.com/ethereum/mist/releases/download/v0.10.0/Ethereum-Wallet-installer-0-10-0.exe (besucht am 03. 06. 2018).

[MistSource] *Mist. Browse and use Đapps on the Ethereum network.* URL: https://github.com/ethereum/mist/ (besucht am 03. 06. 2018).

[MistWiki] *Home · ethereum mist Wiki · GitHub.* URL: https://github.com/ethereum/mist/wiki (besucht am 03. 06. 2018).

[Remix] *Remix Solidity IDE.* URL: http://remix.ethereum.org (besucht am 03. 06. 2018).

[SCBestPractice] *Ethereum Smart Contract Security Best Practices.* URL: https://github.com/ConsenSys/smart-contract-best-practices/ (besucht am 20. 06. 2018).

[Sharding] *Sharding FAQs.* URL: https://github.com/ethereum/wiki/wiki/Sharding-FAQs (besucht am 17. 06. 2018).

[SLOCK.IT] *Slock.it FAQ.* URL: https://slock.it/downloads/slockit_presskit.zip (besucht am 16. 06. 2018).

[Web3.py] *A python interface for interacting with the Ethereum blockchain and ecosystem.* URL: https://github.com/ethereum/web3.py (besucht am 03. 06. 2018).

[Web3Doc] *Web3.py online Dokumentation.* URL: http://web3py.readthedocs.io/en/stable/ (besucht am 03. 06. 2018).

[WHISPER] *Whisper Overview.* URL: https://github.com/ethereum/go-ethereum/wiki/Whisper-Overview (besucht am 16. 06. 2018).

A. Listings zur Simulationsumgebung

Listing A.1: `genesis.json` Datei zur Initialisierung der privaten BC

```
{
    "config": {
        "chainId": 15,
        "homesteadBlock": 0,
        "eip155Block": 0,
        "eip158Block": 0
    },
    "difficulty": "20",
    "gasLimit": "2100000",
    "alloc": {}
}
```

B. Listings zur Simulation des Szenarios 1

Listing B.1: JSON ABI des Smart Contracts `LatestFirmware.sol`.

```
[
        {
                "constant": true,
                "inputs": [],
                "name": "FirmwareVersion",
                "outputs": [
                        {
                                "name": "",
                                "type": "uint16"
                        }
                ],
                "payable": false,
                "stateMutability": "view",
                "type": "function"
        },
        {
                "constant": false,
                "inputs": [],
                "name": "killme",
                "outputs": [],
                "payable": false,
                "stateMutability": "nonpayable",
                "type": "function"
        },
        {
                "constant": true,
                "inputs": [],
                "name": "publisher",
                "outputs": [
                        {
                                "name": "",
                                "type": "address"
                        }
                ],
                "payable": false,
                "stateMutability": "view",
                "type": "function"
        },
        {
                "constant": true,
                "inputs": [],
                "name": "FirmwareHash",
                "outputs": [
                        {
                                "name": "",
                                "type": "string"
                        }
                ],
                "payable": false,
                "stateMutability": "view",
                "type": "function"
```

```
        },
        {
                "constant": false,
                "inputs": [
                        {
                                "name": "fwh",
                                "type": "string"
                        },
                        {
                                "name": "v",
                                "type": "uint16"
                        }
                ],
                "name": "set_LatestFirmware",
                "outputs": [],
                "payable": false,
                "stateMutability": "nonpayable",
                "type": "function"
        },
        {
                "inputs": [],
                "payable": false,
                "stateMutability": "nonpayable",
                "type": "constructor"
        }
]
```

C. Listings zur Simulation des Szenarios 2

Listing C.1: Smart Contract `Sensorwarner7.sol`

```
pragma solidity ^0.4.20;


contract Sensorwarner7 {
    address public owner;              // Vertragseigentuemer
    uint public level;                 // kritische Messwerte
    uint public refund;                // Rueckverguetung
    uint[4] internal sensorsInRegion;  // Anzahl Sensoren pro
       Region
    uint public warnLevel;             // Prozent Meldungen
       pro Region
    uint[4] internal numToCheck;       // Notwendige Anzahl
       Meldungen in der Region fuer Warnung
    unint[4] internal warnNext;

    // Datenstruktur je angemeldetem Sensor
    struct SensorAccount {
        uint region;                   // Region des sensors
        bool isRegistered;
        bool isBlacklisted;
        uint contributions;            // Anzahl gesendeter
           Messungen
        uint lastMeasureTime;          // Zeitpunkt letzte
           Meldung
    }

    /* Verwaltung der registrierte Sensoren */
    mapping (address => SensorAccount) public sensors;

    // Datenstruktur je Messung
    struct Measure {
        uint measurement;              // Messwert
        uint time;                     // Zeitpunkt (des
           Blocks)
    }

    // Array zur Speicherung der Messungen je Region
    Measure[][4] measurements;

    // Events
    event MeldungErhalten(address indexed from, uint Meldung,
       uint Region, uint numTotal, uint numToCheck, uint
```

```
    Meldungsintervall);
event RegistrierungErhalten(address indexed from, uint
    Region);
event Warning(uint region, uint numOfMeasures, uint wlevel
    );


// Konstruktor
function Sensorwarner7() public {
    owner = msg.sender;
    refund = 5000000000000000;
    level = 70;
    warnLevel = 40;
}

// Setter zur Aenderung des Levels für kritische Messwerte
function set_level(uint _level) public {
    require(owner == msg.sender);
    level = _level;
}

// Setter zur Aenderung der Rueckverguetung
function set_refund(uint _refund) public {
    require(owner == msg.sender);
    refund = _refund;
}

/* Registrierungsanfrage */
function register(uint _region) public {
    require (!sensors[msg.sender].isRegistered); // Sensor
        ist noch nicht registriert
    sensors[msg.sender].region = _region;
    sensors[msg.sender].isBlacklisted = false;
    sensors[msg.sender].isRegistered = true;
    sensors[msg.sender].contributions = 0;
    sensors[msg.sender].lastMeasureTime = 0;

    sensorsInRegion[_region]++;

    // Anpassung der notwendigen Anzahl der Meldungen in
       Abhaengigkeit von den
    // Sensoren in der Region
    if (100 * numToCheck[_region] < sensorsInRegion[
       _region] * warnLevel) {
      numToCheck[_region]++;
    }

    // Rueckverguetung
    msg.sender.transfer(refund);
    // Event
    emit RegistrierungErhalten(msg.sender, _region);
}


// Erhaltene Meldung
function putMeasure(uint measure) public {

    require (sensors[msg.sender].isRegistered);
```

```
    require (measure >= level);
    require (!sensors[msg.sender].isBlacklisted);


    // spammer blacklisten
    if (now - sensors[msg.sender].lastMeasureTime < 23
       minutes) {
        sensors[msg.sender].isBlacklisted = true;
        return;
    }

    sensors[msg.sender].lastMeasureTime = now;

    uint r = sensors[msg.sender].region;
    uint l = measurements[r].length;

    // Array um 1 verlaengern fuer den zusaetzlichen
       Messwert
    l++;
    measurements[r].length = l;
    measurements[r][l-1].measurement = measure;
    measurements[r][l-1].time = now;

    // Meldung des Sensors vermerken
    sensors[msg.sender].contributions++;

    // Rueckverguetung
    msg.sender.transfer(refund);

    // Pruefe, ob Anzahl kritischer Meldungen fuer
    // Warnung im Zeitfenster (hier: 5 simulierte Stunden)
        ueberschritten ist
    uint intervall = 0;
    if (numToCheck[r] <= l) {
        intervall = (now - measurements[r][l - numToCheck[
           r]].time);
        if ((intervall < 5 minutes) && (warnNext[r] < now)
           ) {
            // Event zur Warnung
            emit Warning(r, numToCheck[r], 3);
            warnNext[r] = now + 5 minutes;
        }
    }

    // Event zur Meldung
    emit MeldungErhalten(msg.sender, measure, r, l,
       numToCheck[r], intervall);
}

// notwendige Funktion, damit Contract Zahlungen erhalten
   kann
// (die er benoetigt, um Rueckverguetungen taetigen zu
   koennen)
function () public payable {
}

// Destruktor
function kill() public { //self-destruct function,
```

```
        require(owner == msg.sender);
        selfdestruct(owner);
    }
}
```

Listing C.2: JSON ABI zum Smart Contract `Sensorwarner7.sol`

```
[
        { "constant": false, "inputs": [
                { "name": "_level", "type": "uint256" } ],
                "name": "set_level", "outputs": [],
                "payable": false, "stateMutability": "
                   nonpayable", "type": "function" },
        { "constant": false, "inputs": [], "name": "kill", "
           outputs": [],
                "payable": false, "stateMutability": "
                   nonpayable", "type": "function" },
        { "constant": true, "inputs": [], "name": "refund", "
           outputs": [
                { "name": "", "type": "uint256", "value":
                   "5000000000000000" } ],
                "payable": false, "stateMutability": "view", "
                   type": "function" },
        { "constant": false, "inputs": [
                { "name": "measure", "type": "uint256" } ],
                "name": "putMeasure", "outputs": [],
                "payable": false, "stateMutability": "
                   nonpayable", "type": "function" },
        { "constant": true, "inputs": [], "name": "level", "
           outputs": [
                { "name": "", "type": "uint256", "value": "70"
                    } ],
                "payable": false, "stateMutability": "view", "
                   type": "function" },
        { "constant": true, "inputs": [], "name": "owner", "
           outputs": [
                { "name": "", "type": "address", "value": "0
                   x24daaa6d629f705a8295c52208fb3bff78d86738"
                   } ],
                "payable": false, "stateMutability": "view", "
                   type": "function" },
        { "constant": false, "inputs": [
                { "name": "_refund", "type": "uint256" } ], "
                   name": "set_refund", "outputs": [],
                "payable": false, "stateMutability": "
                   nonpayable", "type": "function" },
        { "constant": true, "inputs": [], "name": "warnLevel",
            "outputs": [
                { "name": "", "type": "uint256", "value": "40"
                    } ],
                "payable": false, "stateMutability": "view", "
                   type": "function" },
        { "constant": true, "inputs": [ { "name": "", "type":
           "address" } ], "name": "sensors",
                "outputs": [ { "name": "region", "type": "
                   uint256", "value": "0" },
        { "name": "isRegistered", "type": "bool", "value":
```

```
            false },
                { "name": "isBlacklisted", "type": "bool", "
                   value": false },
                { "name": "contributions", "type": "uint256",
                   "value": "0" },
                { "name": "lastMeasureTime", "type": "uint256
                   ", "value": "0" } ],
                "payable": false, "stateMutability": "view", "
                   type": "function" },
        { "constant": false, "inputs": [ { "name": "_region",
           "type": "uint256" } ], "name": "register",
                "outputs": [], "payable": false, "
                   stateMutability": "nonpayable", "type": "
                   function" },
        { "inputs": [], "payable": false, "stateMutability": "
           nonpayable", "type": "constructor" },
        { "payable": true, "stateMutability": "payable", "type
           ": "fallback" },
        { "anonymous": false, "inputs": [
                { "indexed": true, "name": "from", "type": "
                   address" },
                { "indexed": false, "name": "Meldung", "type":
                    "uint256" },
                { "indexed": false, "name": "Region", "type":
                   "uint256" },
                { "indexed": false, "name": "numTotal", "type
                   ": "uint256" },
                { "indexed": false, "name": "numToCheck", "
                   type": "uint256" },
                { "indexed": false, "name": "Meldungsintervall
                   ", "type": "uint256" } ],
                "name": "MeldungErhalten", "type": "event" },
        { "anonymous": false, "inputs": [
                { "indexed": true, "name": "from", "type": "
                   address" },
                { "indexed": false, "name": "Region", "type":
                   "uint256" } ],
                "name": "RegistrierungErhalten", "type": "
                   event" },
        { "anonymous": false, "inputs": [
                { "indexed": false, "name": "region", "type":
                   "uint256" },
                { "indexed": false, "name": "numOfMeasures", "
                   type": "uint256" },
                { "indexed": false, "name": "wlevel", "type":
                   "uint256" } ],
                "name": "Warning", "type": "event" }
]
```

Listing C.3: Python Skript gen_accounts.py

```
# Python3 Skript zur Generierung von Sensoraccounts
from web3 import Web3, HTTPProvider
import json

sensors = {}
RPCinterface = 'http://192.168.1.107:8545'

# Verbindung zur BC aufbauen
w3 = Web3(HTTPProvider(RPCinterface))

for i in range(1, 21):
   sensor_address = w3.personal.newAccount('test123456')
   sensors["sensor"+str(i)] = sensor_address
   print("generated␣sensor␣account␣" + str(i))

with open('sensor_accounts.json', 'w') as sensor_account_file:
   json.dump(sensors, sensor_account_file)

print("done")
```

Listing C.4: Python Skript initial_ether.py

```
# Python3 Skript, um sensor_accounts mit initialem Ether
   auszustatten
from web3 import Web3, HTTPProvider
import json

RPCinterface = 'http://192.168.1.107:8545'

# Verbindung zur BC aufbauen
w3 = Web3(HTTPProvider(RPCinterface))

# coinbase Account freischalten
w3.personal.unlockAccount(w3.eth.coinbase, 'test123456')

# Sensor Account Adressen auslesen
with open('sensor_accounts.json', 'r') as sensor_account_file:
  sensors = json.load(sensor_account_file)

# Sensor Accounts mit 1 ether vorladen
for sensor in sensors:
  transaction = {"from": w3.eth.coinbase, "to": sensors[sensor
     ], "gas": 900000, "value": w3.toWei(1, "ether")}
  t = w3.eth.sendTransaction(transaction)
  print("lade␣" + sensor + '␣an␣Adresse␣' + sensors[sensor])
```

Listing C.5: Python Skript `register_sensors.py`

```
# Python3 Skript zur Registrierung der Sensoren
from web3 import Web3, HTTPProvider
import json

RPCinterface = 'http://192.168.1.107:8545'
contract_address = '0x2C273a3c9802244D16dcDaBDF1309aD28458a777
   '
abi_file = 'sensorwarner_abi.json'

# Verbindung zur BC aufbauen
w3 = Web3(HTTPProvider(RPCinterface))

# contract abi initialisieren
with open(abi_file, 'r') as abi_definition:
  contract_abi = json.load(abi_definition)

contract = w3.eth.contract(address = contract_address, abi =
   contract_abi)

# Sensor Account Adressen auslesen
with open('sensor_accounts.json', 'r') as sensor_account_file:
  sensors = json.load(sensor_account_file)

# Sensoren registrieren, gleichmaessig verteilt auf die
   Regionen
for sensor in sensors:
  region = int(sensor[6:])%4
  w3.personal.unlockAccount(sensors[sensor], 'test123456')
  transaction = {"from": sensors[sensor], "gas": 900000}
  contract.functions.register(region).transact(transaction)
  print('{}␣registriert␣für␣Region␣{}'.format(sensor, region))
```

Listing C.6: Python Skript `sim_sensor.py`

```
# Python3 Script zur Simulation eines Sensors

import sys
from web3 import Web3, HTTPProvider
import json
import time
import random

# logging nach stdout, daher eprint nach stderr für
   Statusmeldungen in der Konsole
def eprint(* args, **kwargs):
  print(*args, file=sys.stderr, **kwargs)

sensor_address = sys.argv[1]
region = int(sys.argv[2])

eprint('Running␣simulation␣for␣{}␣in␣region␣{}'.format(
   sensor_address, region))

contract_address = '0x2C273a3c9802244D16dcDaBDF1309aD28458a777
   '
```

```
RPCinterface = 'http://192.168.1.107:8545'
abi_file = 'sensorwarner7_abi.json'

# Verbindung zur BC aufbauen
w3 = Web3(HTTPProvider(RPCinterface))

# contract abi initialisieren
with open(abi_file, 'r') as abi_definition:
  contract_abi = json.load(abi_definition)

contract = w3.eth.contract(address = contract_address, abi =
   contract_abi)

# contract state variablen abrufen
level = contract.functions.level().call()

# zwei Tage simulieren
letzte_meldung = 0
gesendet = False
for i in range(48):
  # ca. jede simulierte Stunde einen Messwert
  time.sleep(60 + random.randint(-5, 5))
  t = time.localtime(time.time())
  m = (60 * t.tm_hour + t.tm_min - 21) % 24

  # Messwerte immer unkritisch, außer zur Zeit m % 24 - 4 ==
     region, dann kritische Werte
  if m == region:
    # in 80% der Faelle Messung kritisch
    messung = level + 3 + random.randint(-5,5)
  else:
    # in 20% der Faelle  ist Messung kritisch
    messung = level - 4 + random.randint(-5,5)

  # sende kritische Messung, maximal 1 x pro 24h
  eprint('Zeit␣{},␣letzte␣Meldung␣{},␣Differenz␣{}'.format(
     time.time(), letzte_meldung, time.time() - letzte_meldung
     ))
  if (messung >= level) and (time.time() - letzte_meldung > 23
      * 60):
    transaction = {"from": sensor_address, "gas": 900000}
    w3.personal.unlockAccount(sensor_address, 'test123456')
    contract.functions.putMeasure(messung).transact(
       transaction)
    letzte_meldung = time.time()
    gesendet = True

  print("{},␣{},␣{},␣{},␣{},␣{},␣{},␣{}".format(t.tm_hour, t.
     tm_min, t.tm_sec, m, messung, sensor_address, region,
     gesendet))
  eprint("{},␣{},␣{},␣{},␣{},␣{},␣{},␣{}".format(t.tm_hour, t.
     tm_min, t.tm_sec, m, messung, sensor_address, region,
     gesendet))
  gesendet = False
```

Listing C.7: Python Skript `parallel.py`

```
# Python3 Skript zum verzögerten Start 20 paralleler
   Sensorsimulatoren-Threads

from subprocess import run
import json

# Sensor Account Adressen auslesen
with open('sensor_accounts.json', 'r') as sensor_account_file:
  sensors = json.load(sensor_account_file)

# starte parallele Threads
for sensor in sensors:
  region = int(sensor[6:])%4
  logfile = sensor + ".log"
  print('running:␣python3␣sim_sensor.py␣{}␣{}␣>{}␣&'.format(
      sensors[sensor], region, logfile))
  run('python3␣sim_sensor.py␣{}␣{}␣>{}␣&'.format(sensors[
      sensor], region, logfile), shell=True)
```

Listing C.8: Python Skript `eventlog.py`

```
# Python3 Skript zum Abruf der Contract Events
from web3 import Web3, HTTPProvider
import json
import time

RPCinterface = 'http://192.168.1.107:8545'
contract_address = '0x2C273a3c9802244D16dcDaBDF1309aD28458a777
   '
abi_file = 'sensorwarner_abi.json'

# Verbindung zur BC aufbauen
w3 = Web3(HTTPProvider(RPCinterface))

# contract abi initialisieren
with open(abi_file, 'r') as abi_definition:
  contract_abi = json.load(abi_definition)

contract = w3.eth.contract(address = contract_address, abi =
   contract_abi)

# Filter für die drei unterscheidlichen Events generieren,
   Events ueber alle Bloecke je Filter abrufen
eventfilter1 = contract.events.RegistrierungErhalten.
   createFilter(fromBlock=1, toBlock='latest')
events1 = eventfilter1.get_all_entries()
eventfilter2 = contract.events.MeldungErhalten.createFilter(
   fromBlock=1, toBlock='latest')
events2 = eventfilter2.get_all_entries()
eventfilter3 = contract.events.Warning.createFilter(fromBlock
   =1, toBlock='latest')
events3 = eventfilter3.get_all_entries()

# Events ins Eventlog schreiben
with open('event.log', 'w') as eventlog:
```

```
  for event in events1:
    ev = event['event']
    e = event['args']
    b = event['blockNumber']
    t = w3.eth.getBlock(b)['timestamp']
    timestamp = time.asctime(time.localtime(t))
    eventlog.write('{},␣Block:␣{},␣Event:␣{},␣Details:␣{}\n'.
       format(timestamp, b, ev, e))
  for event in events2:
    ev = event['event']
    e = event['args']
    b = event['blockNumber']
    t = w3.eth.getBlock(b)['timestamp']
    timestamp = time.asctime(time.localtime(t))
    eventlog.write('{},␣Block:␣{},␣Event:␣{},␣Details:␣{}\n'.
       format(timestamp, b, ev, e))
  for event in events3:
    ev = event['event']
    e = event['args']
    b = event['blockNumber']
    t = w3.eth.getBlock(b)['timestamp']
    timestamp = time.asctime(time.localtime(t))
    eventlog.write('{},␣Block:␣{},␣Event:␣{},␣Details:␣{}\n'.
       format(timestamp, b, ev, e))
```

Listing C.9: Python Skript `get_sensor_status.py`

```
# Python3 Script zum Abruf des Sensorstatus

import sys
from web3 import Web3, HTTPProvider
import json

contract_address = '0x6345037d2D72FCBda79FF31C3ED2995900bFA4aF
   '
RPCinterface = 'http://192.168.1.107:8545'
abi_file = 'sensorwarner7_abi.json'

# Verbindung zur BC aufbauen
w3 = Web3(HTTPProvider(RPCinterface))

# contract abi initialisieren
with open(abi_file, 'r') as abi_definition:
  contract_abi = json.load(abi_definition)

contract = w3.eth.contract(address = contract_address, abi =
   contract_abi)

# Sensor Account Adressen auslesen
with open('sensor_accounts.json', 'r') as sensor_account_file:
  sensors = json.load(sensor_account_file)

# Status abrufen und ausgeben
for sensor in sensors:
  sensorstatus = contract.functions.sensors(sensors[sensor]).
     call()
  print('\nSensorstatus␣for␣{}␣@␣{}:\n
```

```
    ----------------------------------------------'.format(
    sensor, sensors[sensor]))
  print('Region:␣{}\nisRegistered:␣{}\nisBlacklisted:␣{}\
    nAnzahl␣Meldungen:␣{}'.format(sensorstatus[0],
    sensorstatus[1], sensorstatus[2], sensorstatus[3]))
```

Listing C.10: Datei `sensor_accounts.json`

```
{
"sensor18": "0x8Ea5A49b5B2042B581A84a61ab727522811ffBd7",
 "sensor5": "0x64F5947fBAb008689BD93acd9709E0396657656a",
 "sensor2": "0x87D0a11D8968B6D065cDe3C239d79eDa97DD8981",
 "sensor17": "0x0fB265216640286115707bF089089Db93CC4948a",
 "sensor4": "0x448923D3CC1724Db2495D250Dc28834d5734228D",
 "sensor14": "0xf5AA4E7F3a594948F8cE6869D2C3f5e9Cf89f7b7",
 "sensor6": "0xC5326a9E74AaF27fD62460a9843afD4403B6cA62",
 "sensor13": "0xBCda938C17586501D5e59fC57c4FEE83B0B05623",
 "sensor3": "0xB40a3C2d9D12e6Ee9010e9a9eB7f47b1AE7c2396",
 "sensor16": "0xfbB942F30023CBc09b6b8E4a3778028c49ff78fB",
 "sensor7": "0xFaA3A4021059B1E025d8419f25EB92C21D72b946",
 "sensor20": "0xA4311Ec9A405a7345Cc310b07401B3e9c82040ce",
 "sensor12": "0x27bf892f566AAE03E5Ff328FB0a26E4D51C178f6",
 "sensor1": "0xaCa9B5ca76EB95e37c07B2C88BBbf3954DA031aD",
 "sensor9": "0x30d5D0c7AE3105bc92B034743fFb8614d0bF1518",
 "sensor10": "0xa2A216095ab03dB2AC0A862180e61EFDf5035a0c",
 "sensor11": "0x5a277bfcbDec631853f36FECAED005BDae2D94F2",
 "sensor15": "0x638EfCfd4D237702b7A99d1F54b64108d692F29b",
 "sensor8": "0x5A993f8f9697b2a98ADeD9Db00DAd6B8a02317B6",
 "sensor19": "0x2F02A5323d7eD06f489424705471c79A76b854af"
}
```

Listing C.11: Sensorstatus nach Beendigung der Simulation (Output von `get_sensor_status.py`)

```
dn@dn-VirtualBox:~$ python3 get_sensor_status.py

Sensorstatus for sensor1
@ 0xaCa9B5ca76EB95e37c07B2C88BBbf3954DA031aD:
----------------------------------------------
Region: 1
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2

Sensorstatus for sensor3
@ 0xB40a3C2d9D12e6Ee9010e9a9eB7f47b1AE7c2396:
----------------------------------------------
Region: 3
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2

Sensorstatus for sensor13
@ 0xBCda938C17586501D5e59fC57c4FEE83B0B05623:
----------------------------------------------
Region: 1
```

```
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2

Sensorstatus for sensor15
@ 0x638EfCfd4D237702b7A99d1F54b64108d692F29b:
----------------------------------------------
Region: 3
isRegistered: True
isBlacklisted: True
Anzahl Meldungen: 1

Sensorstatus for sensor5
@ 0x64F5947fBAb008689BD93acd9709E0396657656a:
----------------------------------------------
Region: 1
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2

Sensorstatus for sensor11
@ 0x5a277bfcbDec631853f36FECAED005BDae2D94F2:
----------------------------------------------
Region: 3
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2

Sensorstatus for sensor7
@ 0xFaA3A4021059B1E025d8419f25EB92C21D72b946:
----------------------------------------------
Region: 3
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2

Sensorstatus for sensor9
@ 0x30d5D0c7AE3105bc92B034743fFb8614d0bF1518:
----------------------------------------------
Region: 1
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2

Sensorstatus for sensor20
@ 0xA4311Ec9A405a7345Cc310b07401B3e9c82040ce:
----------------------------------------------
Region: 0
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2

Sensorstatus for sensor2
@ 0x87D0a11D8968B6D065cDe3C239d79eDa97DD8981:
----------------------------------------------
Region: 2
isRegistered: True
isBlacklisted: False
```

```
Anzahl Meldungen: 2

Sensorstatus for sensor10
@ 0xa2A216095ab03dB2AC0A862180e61EFDf5035a0c:
----------------------------------------------
Region: 2
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2

Sensorstatus for sensor6
@ 0xC5326a9E74AaF27fD62460a9843afD4403B6cA62:
----------------------------------------------
Region: 2
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2

Sensorstatus for sensor8
@ 0x5A993f8f9697b2a98ADeD9Db00DAd6B8a02317B6:
----------------------------------------------
Region: 0
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2

Sensorstatus for sensor4
@ 0x448923D3CC1724Db2495D250Dc28834d5734228D:
----------------------------------------------
Region: 0
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2

Sensorstatus for sensor18
@ 0x8Ea5A49b5B2042B581A84a61ab727522811ffBd7:
----------------------------------------------
Region: 2
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2

Sensorstatus for sensor14
@ 0xf5AA4E7F3a594948F8cE6869D2C3f5e9Cf89f7b7:
----------------------------------------------
Region: 2
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2

Sensorstatus for sensor19
@ 0x2F02A5323d7eD06f489424705471c79A76b854af:
----------------------------------------------
Region: 3
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2
```

```
Sensorstatus for sensor16
@ 0xfbB942F30023CBc09b6b8E4a3778028c49ff78fB:
----------------------------------------------
Region: 0
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2

Sensorstatus for sensor12
@ 0x27bf892f566AAE03E5Ff328FB0a26E4D51C178f6:
----------------------------------------------
Region: 0
isRegistered: True
isBlacklisted: False
Anzahl Meldungen: 2

Sensorstatus for sensor17
@ 0x0fB265216640286115707bF089089Db93CC4948a:
----------------------------------------------
Region: 1
isRegistered: True
isBlacklisted: True
Anzahl Meldungen: 1
```

D. Verzeichnis der digitalen Anlagen

Tabelle D.1.: Digitale Anlagen

Datei	Inhalt
`sensors.log`	Ausgabe der Sensorsimulation nach Szenario 2
`event.log`	Ausgabe des Eventlogs nach Szenario 2
`sensorlog2.xlsx`	Manueller Abgleich von Sensorlog und Eventlog

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Es wurden keine anderen als die angegebenen Stellen und Hinweise verwandt.
Alle Quellen, die dem World Wide Web entnommen oder in einer sonstigen digitalen Form verwendet wurden, sind der Arbeit beigefügt. Der Durchführung einer elektronischen Plagiatsprüfung stimme ich hiermit zu. Die eingereichte Datei entspricht der eingereichten Druckfassung.
Die vorliegende Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Wismar, 28. Juni 2018 .....

Abstract

Detlev Noll

Untersuchungen zum Einsatz von Blockchain Technologien für das Internet der Dinge

Das Internet der Dinge (Internet of Things, IoT) verbindet die physische und virtuelle Welt und ist damit Grundlage für eine Vielzahl innovativer Anwendungen. Blockchain Technologien haben das Potential, nicht nur die automatisierte Verarbeitung von Geschäftstransaktionen zu ermöglichen, sondern auch im Kontext Informationssicherheit und Datenschutz – eine der großen Herausforderungen im IoT – einen signifikanten Beitrag zu leisten. Diese Arbeit erörtert zunächst die Basistechnologien und stellt mögliche Anwendungen und konkrete Beispiele vor, die in der Literatur diskutiert werden. Im Rahmen einer Simulation werden zwei ausgewählte Szenarien realisiert, um Erfahrungen in der praktischen Umsetzung zu sammeln. Auf dieser Basis werden Chancen, Risiken und Einschränkungen diskutiert und bewertet. Werden Lösungen für diese Einschränkungen gefunden, dann können Blockchains perspektivisch als generischer zusätzlicher Layer im IoT-Stack neue Geschäftsmodelle ermöglichen und die Informationssicherheit, den Datenschutz und die Unabhängigkeit von einzelnen Dienstleistern im IoT stärken.

Investigations on the Use of Blockchain Technologies for the Internet of Things

The Internet of Things (IoT) connects the physical and virtual worlds and is therefore the basis for a multitude of innovative applications. Blockchain technologies have the potential to enable not only the automated processing of business transactions, but also to make significant contributions in the context of information security and privacy – being one of the major challenges in the IoT. This work elaborates the basic technologies and presents possible applications and specific examples that are discussed in the literature. As part of a simulation, two selected scenarios are realized in order to gain experience in the practical implementation. On this basis, opportunities, risks and limitations are discussed and evaluated. If solutions to these limitations are found, block chains can serve as a generic additional layer to the IoT stack, thus enabling new business models and strengthen the information security, privacy, and independence of individual service providers in the IoT.