FH MÜNSTER
University of Applied Sciences

KRYPTOWÄHRUNGEN UND SMART CONTRACTS

Marius Spancken, Mario Hellenkamp, Christopher Brown, Christian Thiel

Abschlussbericht zum Forschungs- und Entwicklungsprojekt 2015/2016

im Studiengang Master of Science Wirtschaftsinformatik

an der FH Münster

Dozent: Prof. Dr. Claus Grewe

Fachbereich Wirtschaft | Münster School of Business

Abgabe: 12.02.2016

Vorwort

Das Curriculum des Studiengangs Master of Science Wirtschaftsinformatik an der FH Münster beinhaltet das Wahlpflichtmodul „Forschungs- und Entwicklungsprojekt“. Das Modul ermöglicht es Masterstudierenden, sich forschungsorientiert mit innovativen Themen auseinanderzusetzen. Die behandelten Themen werden von Jahr zu Jahr neu gewählt und umfassen sowohl aktuelle Forschungsgebiete als auch Innovationen, die den IT-Markt bzw. die Wirtschaftsinformatik gerade erreichen bzw. noch nicht durchdrungen haben.

Das Forschungs- und Entwicklungsprojekt 2015/2016 zum Thema „Kryptowährungen und Smart Contracts“ startete im September 2015 und lief bis Februar 2016. Das Projektteam umfasste die vier Mitglieder Marius Spancken, Mario Hellenkamp, Christopher Brown und Christian Thiel. Gemeinsam fertigten sie den vorliegenden Abschlussbericht an.

Eines der definierten Projektziele bestand in der Analyse der Einsatzzwecke, Potenziale und Architekturen von Blockchain-Anwendungen. Des Weiteren sollten verschiedene Fragestellungen zum Nutzen in modernen Geschäftsprozessen und zu den technischen Herausforderungen mittels der Entwicklung eigener Prototypen geklärt werden:

- Wie lässt sich eine Blockchain mittels 51-Prozent-Attacke nachträglich doch noch modifizieren?
- Wie sieht eine konkrete Smart-Contract-Implementierung einer verteilten Clearinghouse-Anwendung in Ethereum aus?

Der vorliegende Bericht vermittelt einen guten Überblick über Ansätze, Strukturen, interne Abläufe und Rahmenbedingungen aktueller Blockchain-Implementierungen. Die erzielten Ergebnisse verdeutlichen neben dem Nutzen auch die Besonderheiten und Einschränkungen der Blockchain-Technologie.

Münster, im September 2016

Prof. Dr. Claus Grewe

Markenrechtlicher Hinweis

Die in dieser Arbeit wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenzeichen usw. können auch ohne besondere Kennzeichnung geschützte Marken sein und als solche den gesetzlichen Bestimmungen unterliegen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

AES	Advanced Encryption Standard
API	Application Programming Interface
BPMN	Business Process Model and Notation
DNS	Domain Name System
DSA	Digital Signature Algorithm
ECB	European Central Bank
ECDSA	Elliptic Curve Digital Signature Algorithm
JSON	JavaScript Object Notation
KG	Kontraktgröße
KP	Kontraktpreis
KWG	Kreditwesengesetz
POA	Proof-of-Activity
POB	Proof-of-Burn
POC	Proof-of-Capacity
POS	Proof-of-Stack
POW	Proof-of-Work
RIPEDM	RACE Integrity Primitives Evaluation Message Digest
RPC	Remote Procedure Call
RSA	Rivest, Shamir und Adleman
SHA	Secure Hash Algorithm
TCP	Transmission Control Protocol
TLS	Transport Layer Security
XML	Extensible Markup Language

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

Mit der Veröffentlichung des Whitepapers „Bitcoin: A Peer-to-Peer Electronic Cash System“ von Satoshi Nakamoto im Jahre 2009 startete die erste praktische Umsetzung einer Kryptowährung, dem Bitcoin. Zunächst als Fundament der Kryptowährung Bitcoin entwickelt, führten aufbauende Entwicklungen zu neuen Nutzungsmöglichkeiten der Blockchain als eigenständige Technologie.

Die vorliegende Ausarbeitung im Rahmen des Forschungs- und Entwicklungsprojekts befasst sich mit der Blockchain-Technologie und der darauf aufbauenden Anwendungsmöglichkeiten. Insbesondere werden dabei Kryptowährungen und Smart Contracts untersucht.

Das Ziel der Ausarbeitung ist es, zunächst ein Verständnis für die zugrundeliegende Technologie zu gewinnen. Auf Basis dessen soll ein neuer Geschäftsprozess fachlich ausgearbeitet und prototypisch implementiert werden.

Die Herausforderungen liegen zum einen im frühen Entwicklungsstadium der Technologie und zum anderen in der Verknüpfung der unterschiedlichen Disziplinen wie der verteilten Infrastruktur, der Kryptographie und der wirtschaftlichen Anwendungsmöglichkeit.

In der vorliegenden Arbeit wird zunächst ein grundlegendes Verständnis von Kryptowährungen und Smart Contracts vermittelt. Anschließend wird die Funktionsweise der Blockchain-Technologie detailliert dargestellt. Darauf aufbauend werden die Herausforderungen und die Umsetzung einer prototypischen Entwicklung dokumentiert. Der anschließende Teil geht auf die Thematik von Smart Contracts ein. Dabei wird zuerst ein auf Smart Contract basierender Geschäftsprozess entwickelt, welcher im Rahmen des Projektes prototypisch samt notwendiger Erweiterungen implementiert wird. Abschließend wird ein Fazit zur Thematik gezogen sowie ein mittelfristiger Ausblick gegeben.

2 Grundlagen

Im nachfolgenden Kapitel werden zuerst die grundlegenden Eigenschaften von Kryptowährungen und Smart Contracts beschrieben, um eine Wissensbasis für die Ausarbeitung zu schaffen. Abschließend erfolgt eine rechtliche Einordnung der Kryptowährungen.

2.1 Kryptowährungen

Eine Kryptowährung stellt ein **digitales Zahlungsmittel** dar. Dieses Zahlungsmittel als Objekt hat keinen Warenwert, an den es geknüpft ist und klassifiziert sich somit als Fiatgeld (Swan, 2015, S. 10). Die Funktionsweise des digitalen Zahlungssystems wird über Verfahren der Kryptografie sowie ein verteiltes dezentrales Netzwerk sichergestellt. Anstelle einer Münze oder eines bedruckten Geldscheins erhält der Inhaber einer Geldeinheit einen kryptografischen Schlüssel. Dieser Schlüssel repräsentiert den Tauschwert der Geldeinheit. Der Wert der Geldeinheit wird durch die Akzeptanz durch Handelspartner bestimmt. Das Besondere an dieser Form von Zahlungssystemen ist, dass keiner dritten Instanz (z.B. Bank) vertraut werden muss. Die dritte Instanz wird durch die zugrunde liegende **Blockchain-Technologie** ersetzt. Die Blockchain beinhaltet somit die gemeinschaftlich erstellte und für jeden zugängliche Buchführung (Nakamoto, 2009, S. 1). Die Funktionsweise dieser Technologie wird in Kapitel 3 näher betrachtet.

Die erste Kryptowährung mit dem Namen **„Bitcoin“** wurde im Januar 2009 von Satoshi Nakamoto publiziert. Aus der durchgeführten Marktanalyse geht hervor, dass aktuell ca. 750 alternative Kryptowährungen existieren (siehe Anhang A).

Name	Marktkapitalisierung	Verfügbare Menge
1. bitcoin	$ 3.478.489.644,00	14.671.600
2. ripple	$ 176.232.498,00	32.488.247.336
3. litecoin	$ 128.175.127,00	42.516.710
4. Ethereum	$ 49.380.869,00	73.575.185
5. bitshares	$ 15.650.096,00	2.511.953.117

Tabelle 1: Top 5 Kryptowährungen nach Marktkapitalisierung (Coinmarketcap)

In Tabelle 1 sind die fünf größten Kryptowährungen gemessen am Wert der Marktkapitalisierung aufgeführt. Die analysierten Kryptowährungen unterscheiden sich im Wesentlichen anhand der folgenden Kriterien:

- Art der verwendeten Kryptografieverfahren (SHA-2, SHA-3, Scrypt etc.)
- Verfahren zur Sicherung der Blockchain (z.B. Proof-of-Work, Proof-of-Stake etc.)
- maximalen Menge an Geldeinheiten (Coins)
- Zeitspanne, in der neue Blöcke generiert werden

Neue Einheiten einer Kryptowährung werden durch **Mining** erzeugt. Das Mining ersetzt die zentrale Institution (z.B. Zentralbank), die sonst neue Einheiten einer Währung ausgibt. Um in den Besitz einer Kryptowährungseinheit zu kommen, ist neben dem Mining auch der Erwerb über diverse Handelsplattformen (z.B. Bitstamp) möglich. In Kapitel 3.3.3 wird das Prinzip des Mining näher betrachtet. Anstelle eines Kontos tritt eine sogenannte **Wallet Software**. Diese Software speichert die für Transaktionen generierten Schlüsselpaare (ECDSA-Verfahren). Der öffentliche Schlüssel dient hierbei als „Kontonummer“, auf die ein anderer Teilnehmer Einheiten überweisen kann. Wer den passenden privaten Schlüssel hat kann über die erhaltenen Währungseinheiten verfügen. Die Wallet Software kann auf einem lokalen Rechner installiert sein oder über eine Börse als Online-Wallet angeboten werden. Es ist auch möglich sein „Guthaben“ in Form von Schlüsselwertpaaren auszudrucken und somit physisch rechnerunabhängig zu speichern (Nakamoto, 2009).

2.2 Smart Contracts

In den nachfolgenden Unterkapiteln werden Smart Contracts, dezentrale Applikationen, kurz DAPPs, in diesem Kontext definiert sowie deren Möglichkeiten und Einschränkungen beschrieben.

Smart Contract

Die Idee von Smart Contracts geht auf eine wissenschaftliche Arbeit von Nick Szabo aus dem Jahre 1998 zurück. Darin wird erstmals beschrieben, wie vertragliche Beziehungen zwischen Parteien gesichert über Computer-Netzwerke möglich sind. Seiner Definition nach sind Smart Contracts computergesteuerte Transaktionsprotokolle, die die Bedingungen eines Vertrages ausführen. Das Ziel ist, die gebräuchlichen Zustände von Verträgen bzw. Transaktionen, wie z.B. Zahlungsbedingungen und die Durchsetzbarkeit, einzuhalten. Daneben soll ein möglicher gegenseitiger Missbrauch der Vertragsparteien minimiert werden. Des Weiteren kann dadurch eine dritte, vertrauliche Partei unter Umständen entfallen (Szabo, 1998). Eine weitere Definition von Mark S. Miller von 2003 bezeichnet Smart Contracts als Programme, die vertragliche Bedingungen und Vereinbarungen zwischen den Vertragsparteien prüfen und durchsetzen können (Miller & Stiegler, 2003).

Weitere Eigenschaften von Smart Contracts sind, dass sie nicht nur Bedingungen prüfen können, sondern auch im Quellcode definierte Aktionen, wie z.B. finanzielle Transaktionen, ausführen können. Außerdem sind sie dezentral, d.h. sie werden nicht an einer zentralen Stelle von einem Betreiber als Dienst bereitgestellt, sondern werden von vielen unterschiedlichen Teilnehmern im Netzwerk ausgeführt. Dies ist mit der Nutzung der Blockchain-Technologie verbunden (Swan, 2015, S. 16). Aus diesen Eigenschaften ergeben

sich sowohl Vorteile, als auch Einschränkungen bei der Portierung von der analogen zur digitalen Welt, welche in den nachfolgenden Unterkapiteln näher beschrieben werden.

DAPP - dezentrale Applikation

Eine dezentralisierte Applikation, kurz DAPP, ist eine Anwendung, die auf Web-Technologien basiert. So werden DAPPs in JavaScript, HTML5 und CSS geschrieben. Zur Ausführung einer Anwendung werden JavaScript-Applikationsplattformen wie, z.B. Node.js, verwendet. Anders als herkömmliche Web-Anwendungen werden dezentrale Applikationen nicht auf einem zentralen Webserver bereitgestellt, sondern können von jedem Nutzer nach Konfiguration der benötigten Laufzeitumgebung, ausgeführt werden. Im Kontext von Smart Contracts bzw. Ethereum sind DAPPs eine Kombination eines in Web-Sprachen geschriebenen Frontends und eines Smart Contracts als Backend-Komponente (Johnston, et al., 2014).

Nutzen und Möglichkeiten

Nutzen und Möglichkeiten ergeben sich vor allem dadurch, dass es sich bei Smart Contracts um Algorithmen handelt und keine menschliche Interaktion, z.B. zur Prüfung von vertraglich vereinbarten Bedingungen, nötig ist. Dadurch erzielt man zum einen Kostenersparnisse , z.B. keine Anwaltskosten, sondern nur geringe Kosten zum Ausführen der Operationen, als auch Zeitersparnisse, da kein Mensch die Prüfung vornehmen muss, denn diese läuft automatisiert ab (Swan, 2015, S. 16). Durch die Verlagerung von einer zentralen Stelle, welcher die Vertragsparteien zur Prüfung vertrauen müssen, z.B. einem Clearinghouse, hin zur Dezentralität und damit einhergehend, dass Vertrauen bzw. der Konsens auf der Grundlage von Vielen basiert, entfällt ebenso der Mittelsmann, der zum gegenwärtigen Zeitpunkt vor allem bei finanzielle Transaktionen zwischen den Vertragsparteien steht. Der Mittelsmann wird in diesem Fall durch einen Algorithmus ersetzt, der im Quellcode fest definierte Prüfungen zur Abwicklungen von Verträgen durchführt (siehe Definition Smart Contract). Daraus resultiert vor allem eine optimierte Vertragsführung. Anders als in Gesetzgebungen üblich gibt es hier keinen Interpretationsspielraum. Die Prüfungen sind nicht vage gehalten, um den Juristen je nach Situation, einen Entscheidungsspielraum zu geben. Unter Umständen können dadurch Rechtsstreitigkeiten vermieden werden, da die vertraglichen Bedingungen nicht änderbar in einem Smart Contract vorliegen (Swan, 2015, S. 17).

Die Anwendungsfälle liegen derzeit vor allem im Bereich der Finanzbranche. Alle Geschäfte, die auf finanzielle Transaktionen oder vertragliche Vereinbarungen aufbauen, können durch Smart Contracts automatisiert abgewickelt werden. Aus diesem Grunde gehen verstärkt Großbanken und Finanzdienstleister in Kooperation mit Beratungsunternehmen das Thema der Blockchain-Technologie und Smart Contracts an, die neue Möglichkeiten zu eruieren (Honsel, 2015).

Einschränkungen

Zum gegenwärtigen Zeitpunkt ist die Idee von Smart Contracts in der realen Welt noch nicht angekommen. Wie bereits erwähnt, ist Bitcoin als Kryptowährung bereits in der Öffentlichkeit bekannt. Smart Contracts in diesem Sinne noch nicht. Hier scheitert es oft bei der Transformation von der digitalen zur physischen Welt sowie bei der rechtlichen Durchsetzbarkeit von Bedingungen, die im Smart Contract festgelegt werden. Als Übergangslösung kann ein Treuhand-Vertrag helfen, der z.B. erst das Geld freigibt, wenn die Ware eingetroffen ist. Dies ist allerdings nur der einfachste Fall. Anders sieht es vor allem bei der Meldepflicht von börsennotierten Kontrakten wie Derivate aus. Diese müssen einer zentralen Stelle gemeldet werden. Weitere offene rechtliche Fragen, um eine Portierung von analogen Verträgen zu Smart Contracts zu ermöglichen, stehen ebenso im Raum (siehe Kapitel 2.3).

Auch ist aufgrund der jungen Entwicklung keine Standardisierung vorhanden. Die bisherigen Plattformen verfolgen im Kern dieselbe Idee, allerdings mit Eigenimplementierungen. Im Gegensatz zu Kryptowährung-Plattformen hat sich noch keine Smart Contract-Plattform am Markt etablieren können. Allerdings sei hier erwähnt, dass „Ethereum“ nach eigener Einschätzung die größte Entwickler- und Nutzergemeinde besitzt. Auch namhafte Firmen sehen hier ein gewisses Potential. So hat Microsoft erst kürzlich die Blockchain-as-a-Service in „Azure“ aufgenommen. Dieser Dienst basiert ebenfalls auf Ethereum (Gray, 2015).

Ein weiteres Thema ist der Schutz von Verträgen. Smart Contracts sind auf der einen Seite in einem gewissen Maße vor Manipulationen geschützt, allerdings sind jegliche vertraglichen Vereinbarungen und Details öffentlich in der Blockchain vorhanden, sodass jeder Teilnehmer des Netzwerkes diese betrachten kann. Hier könnten mit dem Projekt „HAWK“ erste Ansätze zur Lösung des Problems angestrebt werden (Kosba, Miller, Shi, Wen, & Papamanthou, 2015).

2.3 Rechtliche Einordnung

Kryptowährungen und Smart Contracts tangieren durch ihr Wesen eine Reihe von gesetzlichen Regelungen. Aus diesem Grund erfolgt an dieser Stelle eine kurze rechtliche Einordnung der Kryptowährungen und im Besonderen der Kryptowährung Bitcoin. Die Einordnung bezieht sich explizit nur auf Kryptowährungen, da zum Zeitpunkt der Ausarbeitung keine qualitativen Aussagen zur rechtlichen Einordnung von Smart Contracts vorlagen. Die Einordnung gliedert sich in die Punkte Währungsrecht, Lizenzrecht, Einkommens – und Abgeltungssteuerrecht sowie Umsatzsteuerrecht.

Währungsrecht

In Deutschland ist es, nach §14 des Bundesbankgesetzes, nur der Deutschen Bundesbank erlaubt Banknoten in Umlauf zu bringen. Zusätzlich sagt §14, dass auf Euro lautende

Banknoten das Einzige unbeschränkte gesetzliche Zahlungsmittel sind. Daneben ist es nach §35 verboten Geldzeichen (Marken, Münzen, Scheine oder andere Urkunden, die geeignet sind, im Zahlungsverkehr an Stelle der gesetzlich zugelassenen Münzen oder Banknoten verwendet zu werden) auszugeben oder zur Zahlung zu verwenden. (von Uhruh, 2015, S. 62) Die Bundesanstalt für Finanzdienstleistungsaufsicht (BaFin) klassifiziert Bitcoin aber nicht als Zahlungsmittel, sondern als Finanzinstrumente in der Form von Rechnungseinheiten gemäß § 1 Absatz 11 Satz 1 Kreditwesengesetz (KWG) (Münzer, 2013) . Von daher ist davon auszugehen, dass mit der Schöpfung oder Verwendung von Bitcoin kein Verstoß gegen das Bundesbankgesetz vorliegt.

Lizenzrecht

Seitdem Bitcoin immer populärer und beliebter als Zahlungsmittel wird, haben auch die Gesetzgeber die Notwendigkeit erkannt, das Bitcoin-Systeme überwacht und reguliert werden sollten. Plattformen die Bitcoin-Transaktionen anbieten sollten zum Schutz ihrer Nutzer gegen Betrug und zur Verhinderung von Geldwäsche lizensiert werden. In Deutschland sind Betreiber von Finanzdienstleistungen, nach §32 des Kreditwesengesetzes, dazu verpflichtet eine Erlaubnis der Aufsichtsbehörde einzuholen.

KWG §32 (1) Wer im Inland gewerbsmäßig oder in einem Umfang, der einen in kaufmännischer Weise eingerichteten Geschäftsbetrieb erfordert, Bankgeschäfte betreiben oder Finanzdienstleistungen erbringen will, bedarf der schriftlichen Erlaubnis der Aufsichtsbehörde.

Damit Betreiber unter das Kreditwesengesetz fallen, genügt es schon, dass sie ihre Dienstleistungen deutschen Kunden anbieten, sie benötigen keinen Sitz oder eine Niederlassung in Deutschland. Verstöße gegen das Kreditwesengesetz werden mit Geld- und/oder Freiheitsstrafen geahndet. Das Bitcoin-Plattformen unter das Kreditwesengesetz fallen ist nicht etwa der Tatsache geschuldet, dass der deutsche Gesetzgeber besonders schnell eine Anpassung an das Gesetz vorgenommen hat, sondern vielmehr darin, dass deutsche Gesetze in der Regel sehr weit gefasst und abstrakt gehalten sind. (Boehm & Pesch, 2014, S. 44)

Einkommens- und Abgeltungssteuerrecht

Durch die steigende Verbreitung von Bitcoin sind auch die Steuerbehörden auf die Kryptowährung aufmerksam geworden. Sie haben Interesse daran, dass Einkommen rechtmäßig versteuert wird, welches in Form von Bitcoin erzielt wird. Deutsche Steuerbehörden klassifizieren Bitcoin als Wirtschaftsgut und diese müssen daher entsprechend versteuert werden. Dies bedeutet, dass Spekulationsgewinne, welche innerhalb eines Jahres realisiert werden, voll versteuert werden müssen. Davon ausgenommen ist ein Freibetrag von 600 €. Dies bedeutet auch, dass Spekulationsgewinne mit Bitcoin, anders als

z.B. bei Aktien, Zertifikaten oder Fonds, nicht mit 25 % Abgeltungssteuer belastet werden. (Boehm & Pesch, 2014, S. 45)

Umsatzsteuerrecht

Anders sieht die Betrachtung bei der Umsatzsteuer aus. In Deutschland sind nach Paragraf 1 Absatz 1 Umsätze die Lieferungen und sonstigen Leistungen, die ein Unternehmer im Inland gegen Entgelt im Rahmen seines Unternehmens ausführt, steuerpflichtig. Nach dieser Betrachtungsweise, wäre z.B. der Tausch von Euro (Entgelt) gegen Bitcoin (Wirtschaftsgut) über eine kommerzielle Plattform umsatzsteuerpflichtig. Wohingegen ein reiner Währungstausch wie zum Beispiel Euro gegen britische Pfund von der Umsatzsteuerpflicht befreit ist. Hier entschied das Bundesministerium der Finanzen aber*, dass die bloße Entgeltentrichtung keine Lieferung oder sonstige Leistung im Sinne des §1 Absatz 1 des Umsatzsteuergesetzes ist.* Dies geht aus der Antwort des Bundesministeriums der Finanzen vom 27. September 2013 auf die Anfrage des Bundestagsabgeordneten Frank Schäffler hervor (Koschyk, 2013).

3 Funktionsweise der Blockchain-Technologie

Die Blockchain-Technologie ist die Basis für alle aktuellen Kryptowährungen und Smart-Contract-Ansätze. In der Blockchain werden alle jemals in Bezug auf die Kryptowährung getätigten Transaktionen gespeichert. Ziel dieses Kapitels ist es, die komplexe Funktionsweise umfänglich und anschaulich darzustellen.

Zunächst werden die einzelnen Komponenten, aus denen sich die Blockchain zusammensetzt, erläutert. Daraufhin werden die für die Funktionsweise der Blockchain-Technologie notwendigen kryptografischen Verfahren dargestellt. Im dritten Teil werden die vorgestellten Komponenten in Verbindung gebracht und somit die Logik der Blockchain-Technologie veranschaulicht.

3.1 Komponenten

Damit die in den folgenden Kapiteln beschriebenen Blockchain-Komponenten richtig eingeordnet werden können, gibt dieser Abschnitt einen kurzen Überblick über die verwendeten Begrifflichkeiten. In Abbildung 1 ist eine Blockchain mit ihren Komponenten schematisch dargestellt. Eine Transaktion kann als Überweisung eines Geldbetrags von Person A zu Person B gesehen werden. Diese Transaktion wird verschlüsselt und per Broadcast an alle Teilnehmer des Netzwerks gesendet. Die Teilnehmer können zum einen die Rolle des Clients und zum anderen auch die Rolle des Miners einnehmen (siehe Kapitel 3.3.1). Die im Netzwerk veröffentlichten Transaktionen werden auf Validität geprüft. Welche Eigenschaften eine Transaktion erfüllen muss, um als valide gekennzeichnet zu sein, wird in Kapitel 3.1.1 näher betrachtet.

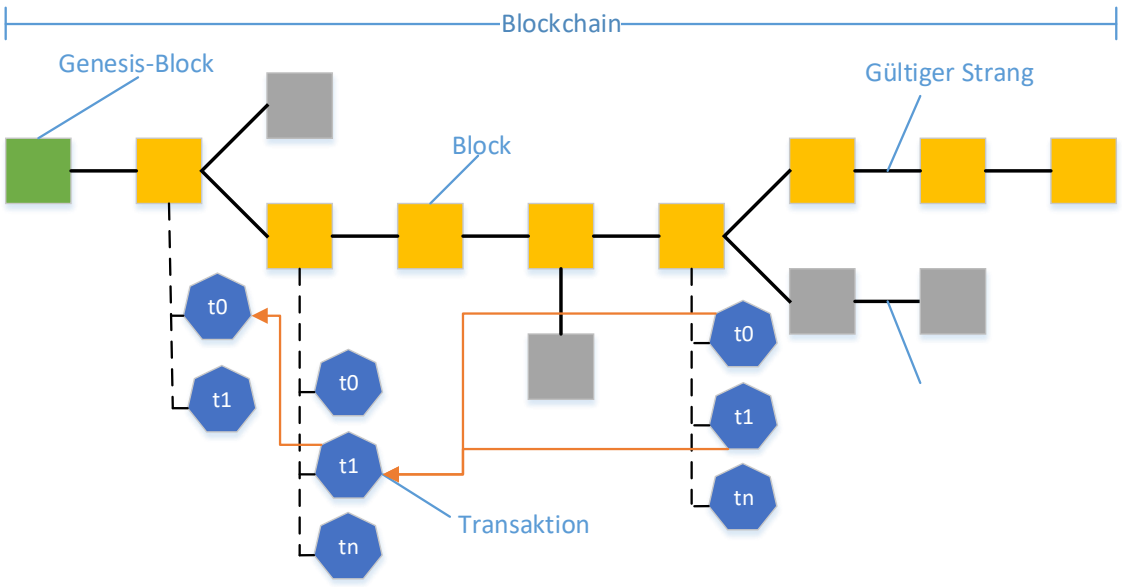

Abbildung 1: Überblick Blockchain-Komponenten

Jeder Teilnehmer in der Rolle des Miners versucht fortlaufend einen neuen validen Block zu generieren und diesen an die aktuell gültige Blockchain anzuhängen. Ein Block umfasst mehrere Transaktionen und kennt seinen Vorgänger. Durch dieses Beziehungswissen entsteht eine Kette von validen Blöcken, die Transaktionen beinhaltet. Der erste Block dieser Kette wird Genesis-Block genannt. Die Transaktionen bestehen aus Inputs und Outputs. Der Ouput einer Transaktion verweist immer auf einen öffentlichen Schlüssel. Der Teilnehmer mit dem passenden privaten Schlüssel kann über den Währungsbetrag, der als Ziel den öffentlichen Schlüssel hat, verfügen, indem er diesen Output als Referenz für seinen Input einer nachfolgenden Transaktion verwendet. So ist es möglich den Geldfluss über die Blockchain bis zu ihrem Ursprung zurück zu verfolgen, da jeder Input genau mit dem einem Output einer vorherigen Transaktion verknüpft ist (Bitcoinwiki-C).

Jeder Teilnehmer des Netzwerks hat eine Kopie der aktuellen Blockchain gespeichert. Es kann vorkommen, dass zwei Teilnehmer in der Rolle des Miners „gleichzeitig“ einen neuen validen Block generieren und diesen im Netzwerk verbreiten. Somit entstehen kurzzeitig parallele Bearbeitungsstränge der Blockchain, die sich im Zeitverlauf wieder korrigieren. Langfristig kann die Blockchain als konsistenter, permanenter und unveränderbarer Speicher gesehen werden (siehe Kapitel 4.3 mit Angriffsszenarien). Die Blockchain nimmt damit die Rolle des Kontobuchs einer Bank ein, indem zu einem Konto alle Transaktionen gespeichert werden (Bitcoin.org-B).

3.1.1 Transaktion

Eine Transaktion ist eine Kernfunktionalität eines Zahlungssystems. Sie ermöglicht das Versenden und das Empfangen von Geldeinheiten. Im Gegensatz zu einer klassischen Transaktion bei einer Bank wird hier nicht die Kontonummer des Empfängers angegeben, sondern sein öffentlicher Schlüssel. Möchte jemand einen Währungsbetrag empfangen, muss dieser sich ein Schlüsselpaar - z.B. über das RSA oder ECDSA Verfahren - generieren und den öffentlichen Schlüssel dem Sender mitteilen. Eine Beispiel zur Schlüsselgenerierung befindet sich in Anhang E.1. Die Transaktion muss dem ACID-Prinzip entsprechen (Vossen & Weikum, 2001, S. 23 - 26):

- **Atomicity**: Ausführung der Transaktion ganz oder gar nicht
- **Consistency**: Wenn der Datenbestand vor der Transaktion konsistent war, muss er es auch im Nachgang sein (Input = Output)
- **Isolation**: Mehrere gleichzeitige Transaktionen dürfen sich nicht beeinflussen
- **Durability**: Die Transaktionen und/oder deren Auswirkungen müssen dauerhaft im System gespeichert sein und können nicht gelöscht werden

In Bezug auf die Blockchain bedeutet dies, dass eine Transaktion die Eigenschaften in einem dezentralen System ohne zentrale Datenbank, ohne zentrale Verarbeitungslogik sowie ohne zentrale Kontrollinstanz einhalten muss. In Abbildung 2 wird der Aufbau einer Transaktion schematisch dargestellt.

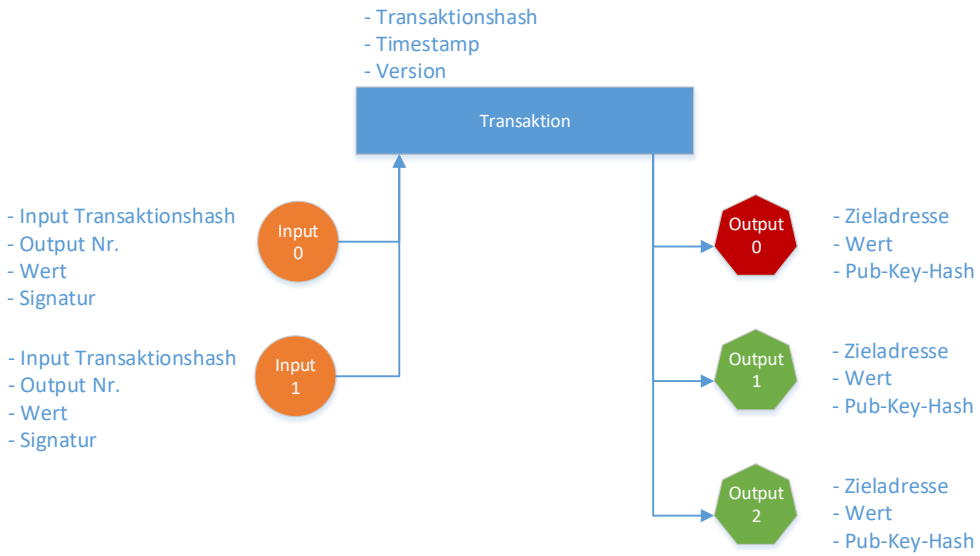

Abbildung 2: Grundlegender Transaktionsaufbau

Eine Transaktion in der Blockchain besteht aus mindestens einem **Input** und mindestens zwei **Outputs**. Die Summe der Werte von Inputs und Outputs muss gleich sein, damit eine Transaktion gültig ist. Ein Input hat immer einen direkten Bezug zu einem Output einer Transaktion, die schon in einem validen Block enthalten ist und noch nicht als Input einer weiteren Transaktion genutzt wurde. Aus diesen Bedingungen entsteht eine eins-zu-eins-Beziehung zwischen Output einer validierten Transaktion und Input einer neuen Transaktion. Diese Beziehung wird über den eindeutigen Transaktionshash und die Output Nr. (also Stelle des Outputs in der Liste der Outputs der vorherigen Transaktion) hergestellt. Über diese Verkettung kann eindeutig bestimmt werden, welche Geldbeträge zu welcher Adresse gesendet wurden. Des Weiteren ist es nicht möglich zwei valide Transaktionen zu erstellen, die den gleichen Output einer vorherigen Transaktion verwenden. Somit ist eine doppelte Ausgabe von Geldbeträgen nicht möglich und somit die Anforderung der „Conistency“ erfüllt (Bitcoinwiki-L, 2015).

Neben den Inputs benötigt eine Transaktion mindestens zwei **Outputs**, wobei der Output mit der Nummer null für die Transaktionsgebühren reserviert ist. Diese **Transaktionsgebühr** wird durch die Differenz von Inputs und Outputs gebildet und an denjenigen ausgezahlt, der den validen Block, welcher die Transaktion beinhaltet, berechnet und an die Blockchain angehängt hat (weitere Informationen hierzu befinden sich im Kapitel 3.1.2.) Jeder Output besteht aus

einer Zieladresse und einem Wert. Die Zieladresse wird aus dem öffentlichen Schlüssel des Empfängers gebildet, vergleichbar mit einer Kontonummer des Empfängers. Der Wert wird vom Sender und/oder von der Software festgelegt. Wie genau eine Transaktion erstellt wird ist in Kapitel 3.3.2 aufgeführt (Bitcoinwiki-L, 2015).

Damit sichergestellt ist, dass nur der Besitzer einer Währungseinheit über diese verfügen kann, wird eine Transaktion vom Transaktionsersteller **signiert**. Hierzu kann das Elliptic Curve Digital Signature Algorithm (ECDSA) Verfahren verwendet werden (Bitcoinwiki-D, 2015). Eine genauere Beschreibung der Funktionsweise von Digitalen Signaturen befindet sich in Kapitel 3.2.1. Die Signierung einer Transaktion wird in der Regel von der Wallet-Software durchgeführt.

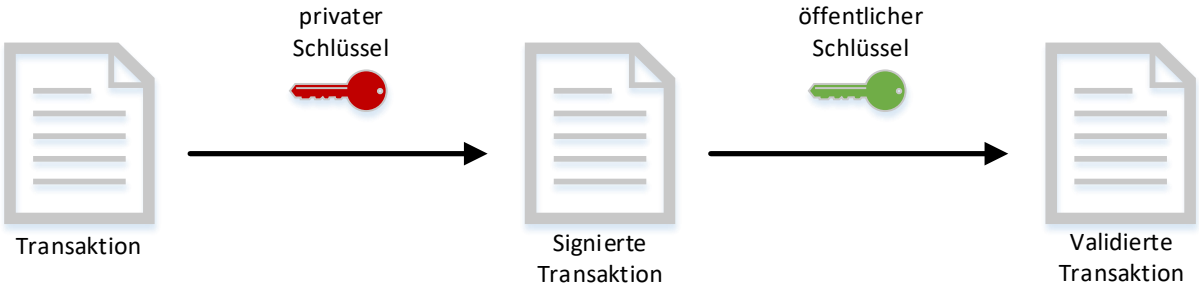

Abbildung 3: Signieren und validieren einer Transaktion

Abbildung 3 zeigt den Zusammenhang von privatem und öffentlichen Schlüssel zu einer Transaktion. Durch die Signierung der Transaktion kann von allen Teilnehmern des Netzwerks über den öffentlichen Schlüssel geprüft werden, ob die Transaktion valide ist und somit vom aktuellen Besitzer der Währungseinheiten kommt. Jeder, der über den privaten Schlüssel verfügt, kann über die an die entsprechende Adresse gesendeten Währungseinheiten verfügen. Ähnlich wie ein Geldschein, der seinen Besitzer nicht kennt und somit von jedem, der ihn physisch besitzt, auch ausgegeben werden kann (Nakamoto, 2009, S. 2).

Aufgrund der dezentralen Datenhaltung ist es notwendig, einen Mechanismus einzuführen, der es unmöglich macht den Inhalt einer Transaktion im Nachgang zu verändern. Zur Lösung dieses Problems wird für die gesamte Transaktion ein **Transaktionshash** gebildet. Dieser beinhaltet alle relevanten Informationen zu einer Transaktion. Werden im Nachgang einzelne Parameter einer Transaktion manipuliert, verändert sich der Transaktionshash und die Transaktion ist nicht mehr valide. Dieser Transaktionshash ist auch für die konsistente Speicherung der Transaktionen in Blöcken und somit in der Blockchain relevant (Nakamoto, 2009, S. 2).

3.1.2 Block

Ein Block stellt eine Sammlung von validen Transaktionen dar, der in der Blockchain persistent gespeichert wird und somit nicht mehr verändert werden kann. In Abbildung 4 sind die relevanten Daten, die ein Block beinhaltet, schematisch dargestellt. Jeder Block kennt seinen Vorgänger, wodurch sich eine Kette von Blöcken bildet. Jeder Teilnehmer in der Rolle des Miners kann den nächsten gültigen Block erstellen und an die Blockchain anhängen. Hierbei ist die Rechenkapazität des Miners relevant, da für die Erstellung eines Blocks ein mathematisches Problem gelöst werden muss. Die Lösung des Problems kann nur durch Trial-and-Error herausgefunden werden. Das Ergebnis der Berechnung ist der gültige Blockhash (Bitcoin.org-A). Eine detaillierte Beschreibung der Generierung eines validen Blocks befindet sich in Kapitel 3.3.3.

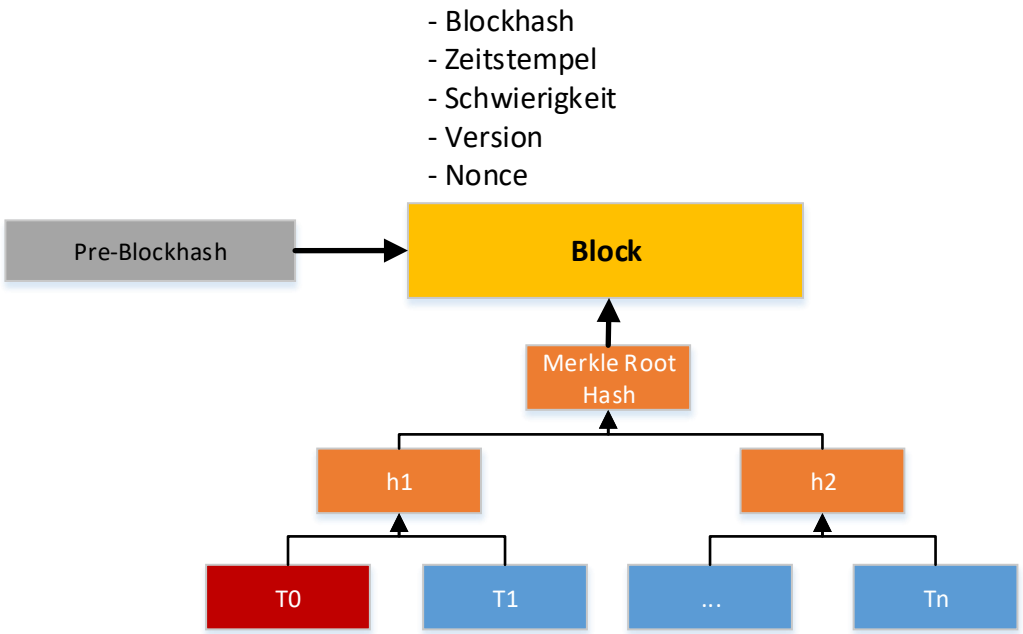

Abbildung 4: Grundlegender Blockaufbau

Welche **Transaktionen in einen Block** aufgenommen werden bestimmt der Miner, der den Block generiert. Dieser hat eine lokale Liste mit verfügbaren Transaktionen, die zuvor per Broadcast an die Teilnehmer des Netzwerks gesendet wurde. Diese Liste wird nach Wert und Transaktionsgebühren sortiert. Somit werden Transaktionen mit einem hohen Wert und somit einer großen Transaktionsgebühr bevorzugt von den Minern berücksichtigt. Die erste Transaktion eines Blocks nennt sich Coinbase-Transaktion, da diese einen besonderen Stellenwert hat. Bei jedem neuen Block wird eine bestimmte Anzahl an Währungseinheiten an den Ersteller des Blocks (Miner) ausgezahlt. Die Menge an Währungseinheiten ist durch die Software festgelegt und reduziert sich im Zeitverlauf. Die Coinbase-Transaktion hat als einzige keinen Input und stellt somit den Ursprung der neuen Währungseinheiten dar. Über die Blockchain kann zu einer Währungseinheit der komplette Transaktionsverlauf bis zur

Coinbase-Transaktion verfolgt werden. Die Transaktionshashes der berücksichtigten Transaktionen werden zu einem Hashbaum zusammengefasst. In Anhang C.1 wird die Berechnung eines solchen Hashbaumes an einem Beispiel erklärt. Durch dieses Verfahren entsteht ein eindeutiger Hashwert, der alle Transaktionen des Blocks berücksichtigt. Wird eine Veränderung an einer Transaktion vorgenommen, entsteht ein komplett anderer Hashwert und der Blockhash ist nicht mehr valide. Somit ist sichergestellt, dass ein Block oder Transaktionen eines Blocks nicht mehr im Nachgang verändert werden können. Neben den Transaktionen wird auch der Bezug zum Vorgängerblock bei der Generierung des Blockhashs berücksichtigt. Eine detaillierte Beschreibung des hier verwendeten Algorithmus (Hashcash) befindet sich in Kapitel 3.3.3. (Bitcoinwiki-C).

Einen validen Block zu generieren ist durch die benötigte mathematische Berechnung sehr aufwändig, wohingegen die Validierung sehr schnell erfolgt. Diese Eigenschaft ist notwendig, da ein erstellter Block durch die Teilnehmer des Netzwerks geprüft und für korrekt befunden werden muss (Bitcoinwiki-B).

3.1.3 Dezentrales Netzwerk/Applikation

Die komplette Anwendung ist dezentral aufgebaut, was bedeutet, dass es keine zentrale Serverinstanz gibt, die Anfragen beantwortet und für die Lauffähigkeit der Software notwendig ist. Jeder, der im Netzwerk mit der Rolle des Clients oder Miners teilnehmen möchte, benötigt lediglich die entsprechende Software. Das Netzwerk basiert auf dem Peer-to-Peer-Prinzip somit kommunizieren die Teilnehmer direkt miteinander. Beim erstmaligen Anmelden am Netzwerk werden über verschiedene DNS-Server aktive Nodes des Netzwerks ermittelt. Mit diesen Nodes werden Verbindungen eröffnet, um weitere Teilnehmer des Netzwerks kennen zu lernen. Beim Erstellen einer Verbindung tauschen die Nodes ihre Liste mit aktiven Nodes aus und ergänzen ihre lokale Liste gegebenenfalls. Die Kommunikation erfolgt auf der TCP-Ebene und stellt somit eine direkte Socket-Kommunikation dar (Decker & Wattenhofer, Information Propagation in the Bitcoin Network, 2013, S. 3 - 4).

In Abbildung 5 sind die zwei Rollen (Miner und Client) dargestellt, die ein Teilnehmer im Blockchain-Netzwerk einnehmen kann. Hierbei ist zu beachten, dass der Miner Rechenkapazität zur Verfügung stellt, um neue Blöcke zu generieren, mit der Intention für das erfolgreiche Erstellen eines Blocks die neu ausgeschütteten Währungseinheiten und die Transaktionsgebühren zu erhalten. Wohingegen der Client seine Wallet verwalten sowie Transaktionen senden und empfangen möchte. Ein Teilnehmer kann die Rolle des Miners und die Rolle des Clients gleichzeitig einnehmen.

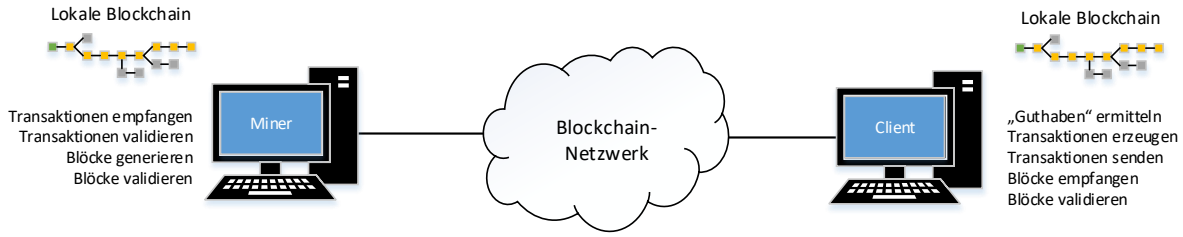

Abbildung 5: Rollen im Blockchain-Netzwerk

Jeder Teilnehmer im Netzwerk hat eine lokale Kopie der Blockchain, die er ständig mit seinen bekannten Nodes angleicht. Des Weiteren hat jeder Teilnehmer eine lokale Liste von bekannten Nodes im Netzwerk, die sich regelmäßig aktualisiert und bei Anfragen erweitert wird. Jeder Miner hat eine lokale Liste von Transaktionen, die sich durch die eingehenden Transaktionen der Clients erweitert. Die Liste wird als Basis für die Blockerstellung genutzt (Bitcoinwiki-G).

Nachdem eine Verbindung mit einer Node erstellt wurde, schickt der Anfragende eine Message mit seinem letzten bekannten Block seiner lokalen Blockchain. Dieser Block wird vom Empfänger mit seinem letzten bekannten Block abgeglichen. Findet der Empfänger „neuere“ Blöcke, sendet er die Liste an Nachfolgeblöcken an den Anfragenden. Dieser holt sich alle fehlenden Blöcke per Anfrage an den Empfänger ab und fügt sie seiner lokalen Liste hinzu. Über diese Methodik aktualisieren die Teilnehmer des Netzwerks ihre lokale Blockchain. Im Netzwerk verbreitet sich automatisch die „längste“ valide Blockchain, da die Nodes sich gegenseitig nach dem letzten validen Block fragen (Bitcoinwiki-F).

Die längste Blockchain wird nicht durch die Anzahl an Blöcken bestimmt, sondern durch den summierten Schwierigkeitsgrad. Dies ist notwendig, um ein Angriffsszenario auszuschließen, bei dem ein Angreifer eine valide Blockchain mit einem sehr niedrigen Schwierigkeitsgrad berechnet und versucht im Netzwerk zu verbreiten (siehe Kapitel Angriffsszenarien 4.3.2). Eine detailliertere Betrachtung des Nachrichtenaustausches zwischen zwei Teilnehmern am Blockchain-Netzwerk wird in Anhang D.1 – D.3 anhand von Sequenzdiagrammen veranschaulicht. In Kapitel 3.3.1 wird näher betrachtet, wie ein Teilnehmer sich innerhalb einer Rolle mit dem Blockchain-Netzwerk verbinden und seine Rolle ausführen kann (Bitcoinwiki-H).

3.2 Grundlage die Kryptografie

Ein wesentlicher Bestandteil der Funktionsweise der Blockchain-Technologie sind die verwendeten Kryptografie Verfahren. Zum einen werden **digitale Signaturen** verwendet um sicherzustellen, dass Transaktionen vom korrekten Absender stammen. Zum anderen kommen verschiedene **Hashfunktionen** zum Einsatz, um Daten vor nachträglicher Veränderung zu schützen.

3.2.1 Digitale Signaturen

Eine digitale Signatur ist eine schlüsselabhängige Prüfsumme. Diese Prüfsumme in Kombination mit einem Dokument (in diesem Fall einer Transaktion) weist ähnliche Eigenschaften auf wie eine Unterschrift (Schmeh, 2013, S. 202):

- Fälschungssicher
- Prüfbarkeit der Echtheit
- Keine Übertragbarkeit zu einem anderen Dokument
- Keine unbemerkte Veränderung des Dokuments

Abbildung 6 illustriert die digitale Signatur am Beispiel einer Transaktion. Für eine digitale Signatur wird ein Schlüsselpaar bestehend aus einem privaten und einem öffentlichen Schlüssel benötigt. Der private Schlüssel in Kombination mit dem zu signierenden Urbild (hier die Transaktion) ergibt die digitale Signatur. Mit Hilfe des öffentlichen Schlüssels kann ein Dritter prüfen, ob das Urbild verändert wurde. Wie die Erstellung eines Schlüsselpaares und die anschließende Prüfung vom Prinzip funktioniert wird in Anhang E.1 an einem Beispiel gezeigt (Schmeh, 2013, S. 202).

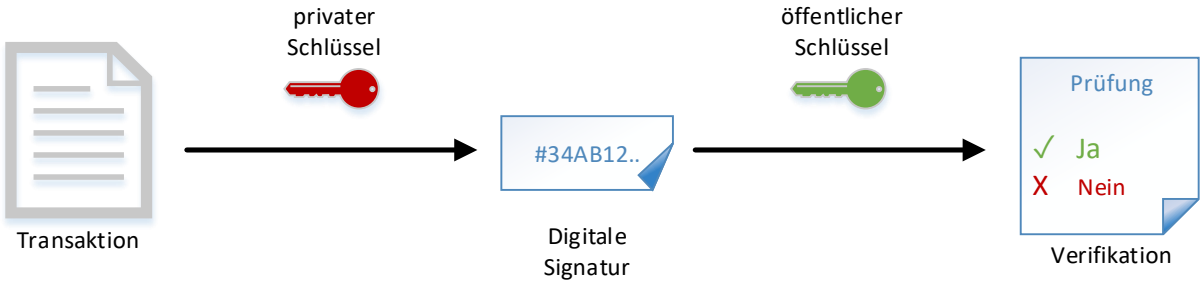

Abbildung 6: Digitale Signatur am Beispiel - Transaktion

Als Verfahren zum Signieren von Transaktionen können zum einen asymmetrische Verschlüsselungsverfahren (wie z.B. RSA) zum anderen Signaturverfahren auf Basis des diskreten Logarithmus (wie z.B. DAS) verwendet werden. Im Fall von Bitcoin wird hierzu die Abwandlung ECDSA eingesetzt. Diese Abwandlung basiert auf elliptischen Kurven und ermöglicht eine kürzere Schlüssellänge bei gleicher Sicherheit (Schmeh, 2013, S. 216).

3.2.2 Kryptografische Hashfunktionen

Hashfunktionen dienen dazu die Prüfsumme für ein Urbild zu erstellen. Das Urbild wird im Fall der Blockchain-Technologie entweder aus Teilen einer Transaktion oder aus Teilen eines Blocks zusammengesetzt. Bei einer Transaktion wird das Urbild aus den Input- (Transaktionshash der vorherigen Transaktion, Wert, Signatur usw.) sowie aus den Output-Daten (Adresse, Wert usw.) gebildet. Sobald eine Änderung an einem Parameter innerhalb der Transaktion durchgeführt wird, ändert sich der Transaktionshash.

Die verwendeten Hashfunktionen sichern die Blöcke und Transaktionen einer Blockchain vor nachträglicher Veränderung. Da die Daten für jeden im Netzwerk transparent vorliegen, kann auch jeder Teilnehmer die Gültigkeit der Hashwerte validieren.

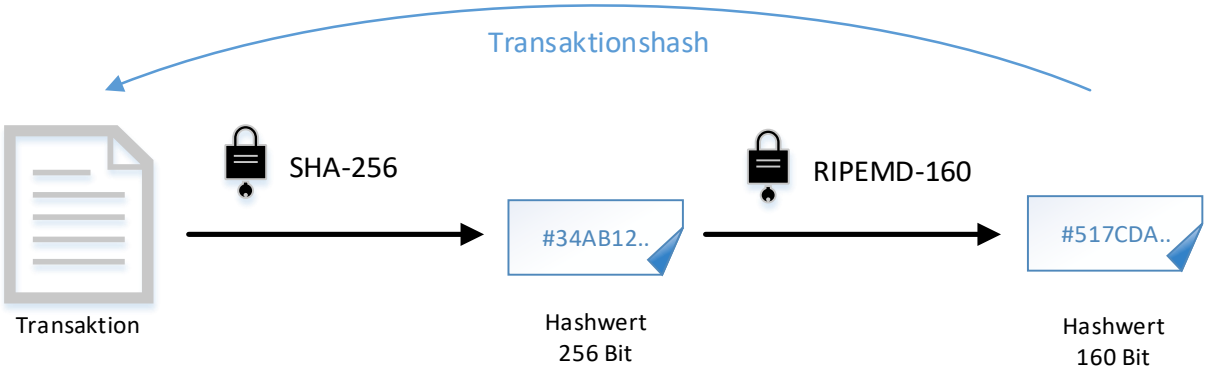

Abbildung 7: Kryptografische Hashfunktion am Beispiel - Transaktion

Aktuell werden die Verfahren SHA-2, SHA-3 und RIPEMD-160 zur Generierung der Hashwerte eingesetzt. Im Fall von Bitcoin wird doppelt gehasht. Wie Abbildung 7 zeigt, werden die Daten zunächst mittels SHA-256 gehasht, worauf der entstehende Hashwert wiederum mittels RIPEMD-160 oder SHA-256 gehasht wird. Ziel dieser doppelten Bearbeitung der Daten ist zum einen der kürzere Hashwert und zum anderen die erhöhte Sicherheit gegen Angriffe. In Anhang E.2 wird das Prinzip von SHA256 anhand eines Schaubilds verdeutlicht.

Es werden grundsätzlich zwei Abstufungen bei den Angriffen auf kryptografische Hashfunktionen unterschieden. Die Suche nach **Kollisionen** ist die Erste. Hierbei wird versucht zwei beliebige Urbilder zu finden, welche den gleiche Hashwert ergeben. Dies wird als Kollision bezeichnet. Das zweite Angriffsszenario ist die Suche nach einem **zweiten Urbild,** wobei ein Urbild oder der Hashwert vorgegeben ist. Im Folgenden werden mögliche Angriffsszenarien aufgeführt und in Zusammenhang mit der Blockchain-Technologie gebracht (Schmeh, 2013, S. 228-236):

- *Substitutionsattacke:* Der Angreifer fängt in diesem Fall die Nachricht inkl. zugehörigem Hashwert ab und versucht durch Ersetzen von Wörtern eine Nachricht zu generieren, die den gleichen Hashwert erzeugt. Je länger in diesem Fall der Hashwert ist, umso mehr Möglichkeiten muss der Angreifer ausprobieren. Bei einer Hashwertlänge von RIPEDM mit einem Wert von 160 Bit müsste der Angreifer 160 Ersetzungsmöglichkeiten finden und daraus 2^{160} Hashwerte bilden.
- *Geburtstagsangriff:* Umfasst der Hashwert eine Länge von 128 Bit gibt es 2^{128} mögliche Hashwerte. Das Geburtstagsproblem besagt, dass der Angreifer im Durchschnitt nur $2^{128/2} = 2^{64}$ Versuche benötigt, um eine Kollision zu finden. Aus diesem Grund benutzen alle neueren Hashverfahren und auch die aktuellen Kryptowährungen mindestens eine Bit Länge von 160.

- *Wörterbuch-Angriff:* Der Wörterbuchangriff nutzt die Tatsache aus, dass z.B. Computersysteme von Passwörtern nur den zugehörigen Hashwert speichern. Somit ist es möglich, mit Hilfe einer entsprechenden Tabelle, die Wörter mit ihrem zugehörigen Hashwert enthält, die Hashwerte zu vergleichen. Wird der gleiche Hashwert gefunden weiß der Angreifer das dazu gehörige Passwort. Dieser Angriff macht in Bezug auf die Blockchain keinen Sinn, da das Urbild (Transaktionen oder Blöcke) öffentlich zugänglich sind.
- *Regenbogentabellen:* Die kryptografischen Hashfunktionen haben die Eigenschaft, dass diese bei einem gleichen Urbild auch immer den gleichen Hashwert erzeugen. Anders als bei Verschlüsselungsverfahren wird kein Schlüssel zur Generierung des Hashwertes verwendet. Somit kann die erfolgreiche Suche nach einer Kollision nicht durch einen Schlüsselwechsel wieder zunichte gemacht werden. Hieraus ergibt sich die Möglichkeit, alle erfolgreichen Angriffe als Urbild-Hashwert-Paar in einer Tabelle zu speichern. Diese Tabelle kann genutzt werden, um zu einem Hashwert das passende Urbild zu finden. Der Vergleich von Hashwerten ist deutlich performanter als die Berechnung. Jedoch benötigt eine solche Tabelle viel Speicherplatz.

Diese Beispiele verdeutlichen die Angriffsmöglichkeiten auf kryptografische Hashfunktionen. Im Gegensatz zu symmetrischen oder asymmetrischen Verschlüsselungsverfahren kann einem Angriff hierbei nur mit einem längeren Hashwert begegnet werden. Die aktuellen Hashverfahren verwenden eine Hashwertlänge zwischen 160 und 512 Bit. Mit größerer Hashwertlänge steigt jedoch auch der Aufwand bei der Berechnung. Somit erzeugt die Hashwertlänge immer ein Trade-off zwischen Aufwand und Sicherheit (Schmeh, 2013, S. 228 - 236).

3.3 Von der Transaktion zum validierten Block

In den Kapiteln 3.1 und 3.2 wurden die Komponenten der Blockchain-Technologie und der Zusammenhang mit der Kryptografie veranschaulicht. Im folgenden Kapitel wird der Prozess von der anfänglichen Verbindung mit dem Blockchain-Netzwerk über die Erstellung und Veröffentlichung einer Transaktion bis zum generierten Block behandelt.

3.3.1 Verbinden mit dem Blockchain-Netzwerk

Für die Erstellung von Transaktionen für ein Blockchain-Netzwerk muss sich der Teilnehmer zunächst mit dem Netzwerk verbinden. Für diesen Zweck gibt es eine dezentrale Software, die jeder Teilnehmer lokal installieren muss. Diese Software stellt die Verbindung zum Netzwerk her und holt sich die benötigten Informationen wie zum Beispiel den aktuellen Stand der Blockchain, Adressen weiterer Nodes oder auch von anderen Teilnehmern gesendete

Transaktionen (nur für das Mining relevant). Diese Software liefert die Basisfunktionalitäten, um sich aktiv am Netzwerk zu beteiligen und wird daher auch Core genannt (Bitcoinwiki-A).

Je nach Rolle, die ein Teilnehmer ausführen möchte, wird von diversen Anbietern Wallet-Software oder Mining-Software angeboten. Die **Wallet-Software** stellt die Geldbörse oder das Konto des Clients dar. Der Unterschied zu einem Bankkonto liegt darin, dass keine Geldbeträge, sondern Schlüsselpaare gespeichert werden. Auf Basis dieser Schlüsselpaare kann die Wallet-Software ein Guthaben aus der Summe der eingehenden Transaktionen bilden. Hierzu nutzt die Wallet Software die Basisfunktionalitäten der Core-Software, um die gesamte Blockchain nach diesen Transaktionen zu durchsuchen. Dieses Vorgehen ist möglich, da die komplette Blockchain öffentlich zugänglich ist und somit von jedem gelesen werden kann. Neben der Anzeige des aktuell verfügbaren Guthabens ist die Wallet-Software für das Erzeugen von Schlüsselpaaren als Adresse für eingehende Transaktionen sowie das Erstellen und Versenden von Transaktionen zuständig. Eine detaillierte Betrachtung zur Erstellung von Transaktionen befindet sich in Kapitel 3.3.2 (Bitcoin.org).

Es gibt unterschiedliche Arten von Wallets, die sich in ihrer Sicherheit und Funktionsweise unterscheiden. Zu den Funktionen einer Wallet zählen die Generierung eines privaten Schlüssels, das Ableiten eines öffentlichen Schlüssels, das Versenden des öffentlichen Schlüssels, das Überwachen der Outputs, das Erstellen von Transaktionen, das Signieren von Transaktionen und das Versenden von Transaktionen. Im Folgenden werden die einzelnen Arten von Wallets kurz aufgeführt (Bitcoin.org):

- *Full-Service Wallet:* Bietet alle Funktionen auf einem Gerät und ist mit dem Netzwerk verbunden (Online-Wallet)
- *Signing-Only Wallet*: Die Signing-Only Wallet übernimmt die Erstellung des privaten Schlüssels und die Signierung von Transaktionen. Sie ist nicht mit dem Netzwerk verbunden (Offline Wallet) und arbeitet mit einer Online Wallet zusammen. Diese Trennung erhöht die Sicherheit. Eine mögliche Erweiterung stellt eine Hardware Wallet dar, die an eine spezielle Hardware gebunden ist und ebenfalls eine erhöhte Sicherheit bietet.
- *Distributing-Only Wallet:* Diese Wallet ist gemacht, um auf schwer zu schützenden Umgebungen (wie z.B. Webserver) zu laufen. Die Wallet ist so konfiguriert, dass nur die Weitergabe von öffentlichen Schlüsseln erlaubt ist.

Je nach Einsatzgebiet und Sicherheitsanspruch kommen unterschiedliche Wallet-Formen infrage. Alle Wallets speichern die erstellten, privaten Schlüssel in einer verschlüsselten Datei (Bitcoin.org).

Möchte ein Teilnehmer die Rolle des Miners einnehmen und seine Rechenkapazität für das Erstellen und Validieren von Blöcken nutzen, wird eine **Mining-Software** benötigt. Auch diese

Software greift auf die Funktionalitäten der Core-Software zurück, um Informationen aus dem Netzwerk zu empfangen und Daten an das Netzwerk zu senden. Der Miner hat die Aufgabe eingehende Transaktionen auf Validität zu prüfen und diese in einem Block zusammenzufassen. Zu diesem erstellten Block muss ein Blockhash berechnet werden, damit dieser Block der Blockchain angehängt werden kann. Der Miner benötigt für das Erstellen eines Blocks Rechenkapazität und wird dafür mit Währungseinheiten belohnt. Diese Währungseinheiten setzen sich aus den summierten Transaktionsgebühren der im Block enthaltenen Transaktionen und einer von der Software festgelegten Ausschüttung neuer Währungseinheiten zusammen (siehe Coinbase-Transaktion in Kapitel 3.1.2). Eine detaillierte Beschreibung des Algorithmus zum Erstellen eines Blocks wird in Kapitel 3.3.3 aufgeführt (Bitcoinwiki-G).

In den Anfängen der Kryptowährungen wurde primär Solo-Mining betrieben, bei dem jeder Teilnehmer alleine versuchte mit seiner Rechenkapazität einen Block zu generieren. Mit steigender Nutzeranzahl und somit auch steigender Rechenkapazität wurde es für den einzelnen Nutzer bei gleicher Rechenkapazität immer unwahrscheinlicher einen validen Block als Erster zu erstellen. Aus diesem Grund bildeten sich Mining-Pools, in denen sich Nutzer mit ihrer Rechenkapazität zusammen schließen, um die Chance, einen validen Block zu erstellen, zu erhöhen (Bitcoinwiki-G).

3.3.2 Erstellung und publizieren einer Transaktion

Möchte ein Teilnehmer einem anderen Teilnehmer einen Währungsbetrag senden oder von einem anderen Teilnehmer einen Währungsbetrag empfangen, müssen beide Parteien als Client mit dem Blockchain-Netzwerk verbunden sein. Der Empfänger generiert z.B. mit dem ECDSA-Verfahren eine Kombination aus öffentlichem und privatem Schlüssel. Der öffentliche Schlüssel wird mittels kryptografischer Hashfunktion (z.B. RIPEMD-160) gehasht. Die entstehende Zeichenfolge hat die Funktion einer Kontonummer. Der Sender bekommt vom Empfänger den gehashten öffentlichen Schlüssel und erstellt eine Transaktion mit der Zeichenfolge als Zieladresse. Neben der Zieladresse und dem Betrag, der gesendet werden soll, wird noch die entsprechende Transaktionsgebühr hinterlegt. Daraufhin ermittelt die Software den benötigten Gesamtbetrag und prüft, ob genügend eingehende Transaktionen zur Verfügung stehen, um den Betrag zu decken. Ist diese Prüfung erfolgreich, ordnet die Software der Transaktion entsprechend viele Inputs zu. Wie bereits zuvor erwähnt, kann ein Output einer Transaktion immer nur eine 1:1 Beziehung zu einem Input einer neuen Transaktion haben. Ein Output kann somit nicht auf zwei Transaktionen aufgeteilt werden. Diese Beziehung zwischen Input und Output verschiedener Transaktionen wird in Anhang B.1 an einem Beispiel verdeutlicht (Bitcoinwiki-I).

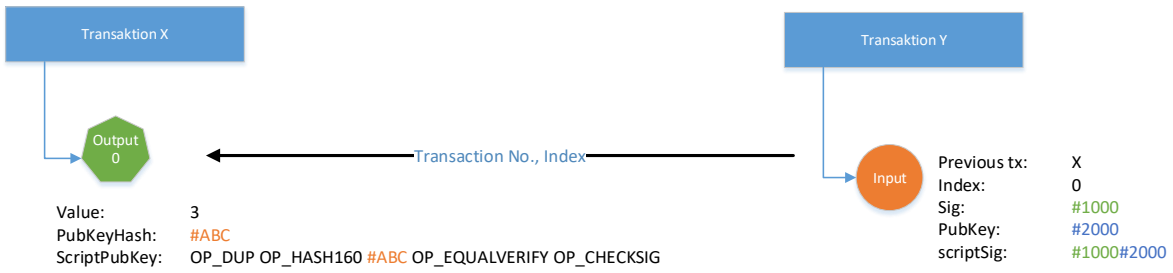

Abbildung 8: Verknüpfung von Input und Output zweier Transaktionen

Abbildung 8 zeigt die Verknüpfung zwischen dem Input einer Transaktion Y zu dem Output einer vorherigen Transaktion X. In diesem Beispiel ist der Wert des Outputs von Transaktion X an die Zieladresse „#ABC“ gesendet worden. Der Besitzer des passenden privaten Schlüssels kann eine Transaktion erstellen, die einen Input hat, der auf den Output der vorherigen Transaktion X verweist. Nur mit dem passenden privaten Schlüssel kann die für den Input benötigte Signatur (in diesem Beispiel #1000) erstellt werden. Wie andere Teilnehmer des Netzwerks die Gültigkeit der Signatur prüfen können, wird in Anhang B.2 weiter ausgeführt (Bitcoinwiki-I).

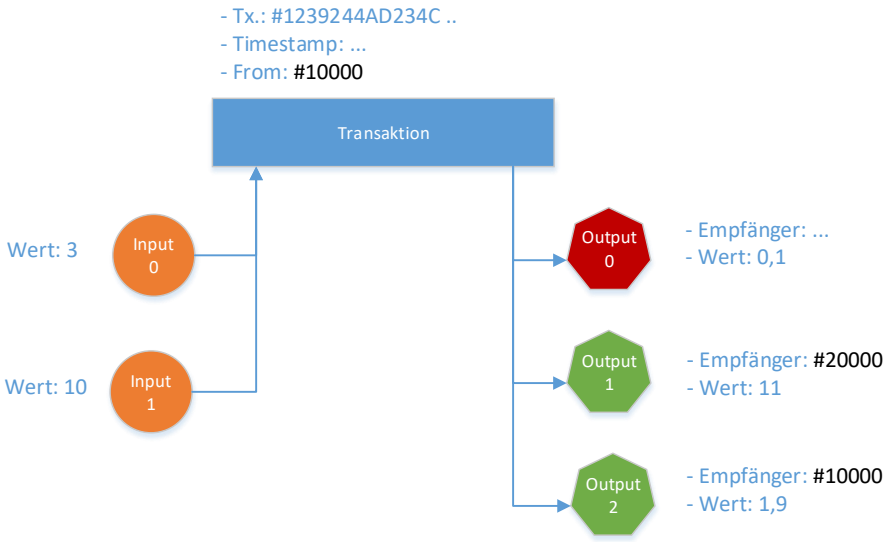

Abbildung 9: Transaktion - Beispiel Input/Output-Verteilung

In Abbildung 9 wird ersichtlich, wie die zuvor beschriebene 1:1-Beziehung von Input und Output die aktuelle Transaktion beeinflusst. Möchte der Teilnehmer #10000 dem Teilnehmer #20000 einen Betrag von 11 Währungseinheiten senden, prüft die Software, ob genügend Inputs vorhanden sind, um auf den Betrag von 11 Einheiten plus 0,1 Einheiten Transaktionsgebühren zu kommen. In diesem Beispiel wurden zwei mögliche Inputs mit den Werten 3 und 10 gefunden. Die möglichen Inputs können jedoch nicht nur zum Teil genutzt werden, da sonst eine 1:n-Beziehung zwischen Input und Output entstehen würde. Somit

entsteht eine Differenz vom benötigten Betrag (11,1 Einheiten) zu einem summierten Input (13 Einheiten). In diesem Fall wird der Transaktion ein weiterer Output mit dem offenen Betrag und der eigenen Zieladresse hinzugefügt. Der Teilnehmer #10000 kann über den Restwert von 1,9 Einheiten verfügen, wohingegen der Empfänger #20000 über die an ihn gesendeten 11 Währungseinheiten verfügen kann (Bitcoinwiki-L, 2015).

Damit die Transaktion im Nachgang nicht verändert werden kann, wird für die Parameter einer Transaktion ein eindeutiger Transaktionshash generiert. In Anhang B.3 wird anhand einer realen Transaktion des Bitcoin-Netzwerks die Erstellung des Transaktionshashs nachvollzogen. Die gesamte Transaktion wird vom Ersteller per Broadcast an alle ihm bekannten Nodes gesendet. Diese Nodes nehmen die Transaktion (falls noch nicht vorhanden) in ihre Liste noch offener Transaktionen auf und versenden diese wiederum an alle ihnen bekannten Nodes weiter. Durch diesen Verteilungsmechanismus ist sichergestellt, dass die Transaktion im Netzwerk bekannt ist und somit in einen der nächsten Blöcke aufgenommen wird. Das Versenden und Empfangen von Transaktionen wird in Anhang D.3 anhand eines Sequenzdiagramms veranschaulicht.

3.3.3 Blockgenerierung und Validierung

In Kapitel 3.1.2 wurde der grundlegende Aufbau eines Blocks definiert. Im Folgenden wird die Erstellung eines validen Blocks näher betrachtet. Die Blockerstellung ist Aufgabe der Miner, die im Netzwerk angemeldet sind. Das Prinzip des mathematischen Problems wurde 1997 von Adam Back entwickelt und ist unter dem Namen Hashcash-Algorithmus bekannt. Das Prinzip wurde ursprünglich zur Reduzierung von Spam-Mails entwickelt. Der Sender einer Mail muss zunächst Rechenkapazität einsetzen, um einen Hashwert passend zu seiner Mail zu berechnen. Das Erstellen eines solchen Hashwertes benötigt mehrere Durchläufe des Algorithmus wohingegen die Prüfung mit nur einer Berechnung durchgeführt werden kann. Somit müsste ein Sender von Massenmails auch einen entsprechend hohen Rechenaufwand für das Erstellen dieser Mails aufbringen, wohingegen der Empfänger nur einen geringen Aufwand bei der Prüfung hat (Back, 2002).

In Bezug auf die Blockgenerierung wurde dieses Prinzip übernommen und angepasst. Jeder Miner stellt sich auf Basis seiner Transaktionsliste einen Block zusammen und versucht zu diesem Block den passenden Blockhash über den Hashcash-Algorithmus zu finden. Dieses Verfahren wir auch Proof-of-Work genannt (Bitcoinwiki-E).

In Abbildung 10 sind die Parameter aufgeführt, die für den Hashcash-Algorithmus benötigt werden. Der Service String setzt sich aus den relevanten Blockdaten wie z.B. dem Vorgänger Blockhash, dem Roothash des Transaktionsbaums, dem aktuellen Schwierigkeitsgrad, einem Zeitstempel und der Versionsnummer zusammen. Dieser Service String umfasst alle Daten,

sodass der Block nicht im Nachgang geändert werden kann ohne einen neuen Proof-of-Work zu finden. Die variablen Teile der Zeichenkette sind die Nonce und der Counter, die jeweils durch einen 32-Bit Integer Wert repräsentiert werden. Die Nonce wird bei jedem Durchlauf des Algorithmus erhöht. Ist die Nonce bei ihrem Maximum (32 Bit = 2.147.483.647) angekommen, startet sie wieder bei null und der Counter wird hochgezählt. Der Counter wird als Nachricht in der Coinbase-Transaktion eingetragen, was zur Folge hat, dass der Merkle Root neu berechnet werden muss (Bitcoinwiki-E).

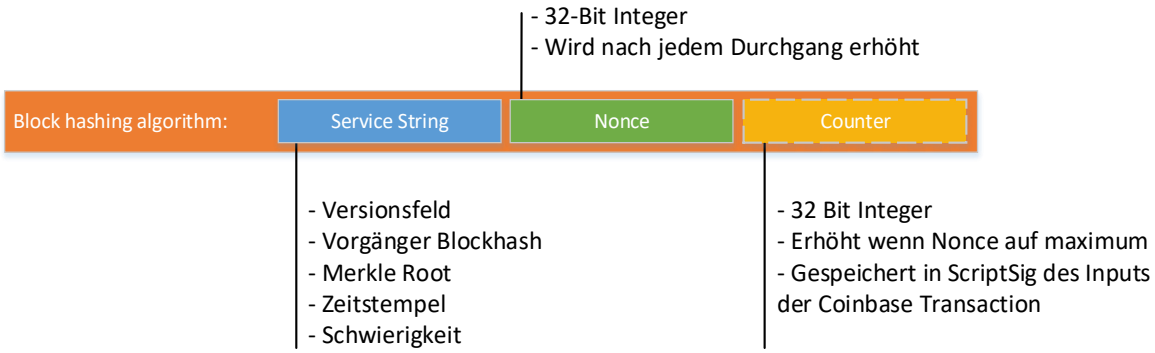

Abbildung 10: Block hashing Algorithmus – Hashcash

Die auf Basis der Parameter entstehende Zeichenfolge wird im Fall von Bitcoin doppelt mit dem kryptografischen Hashverfahren SHA256 gehasht. Der sich ergebende Hashwert muss mit einer, dem Schwierigkeitsgrad entsprechenden, Menge von Nullstellen beginnen. Der Schwierigkeitsgrad wird von der Software in regelmäßigen Abständen an die im Netzwerk verfügbare Rechenkapazität angepasst. In Anhang C.2 wird eine solche Berechnung an einem Beispiel verdeutlicht. Weiterführende Informationen bezüglich des Hashcash-Algorithmus werden in Anhang C.3 aufgeführt (Bitcoinwiki-K).

Hat ein Miner einen validen Blockhash zu seinem Block gefunden, publiziert er diesen per Broadcast im Netzwerk. Die Nodes im Netzwerk prüfen den Block und seinen Blockhash und hängen diesen an ihre lokale Blockchain (sofern alle Parameter stimmen). Ein Beispiel zur Prüfung des Blockhashs anhand eines realen Blocks der Bitcoin Blockchain ist in Anhang C.4 aufgeführt. Da jeder Miner an seiner lokalen Blockchain arbeitet, kann es vorkommen, dass ein Miner einen validen Block nicht empfängt. Ist dies der Fall, hebt sich der Fehler spätestens beim nächsten validen Block, den der Miner bekommt, wieder auf. Um die Veränderung der lokalen Blockchain zu verdeutlichen, wurde ein Schaubild erstellt, welches den Prozess der Blockgenerierung anhand von drei Minern beispielhaft veranschaulicht. Dieses Schaubild befindet sich in Anhang C.5. Stellt der Miner fest, dass ihm ein Block in seiner Kette fehlt, fordert er diesen Vorgängerblock vom Sender des neuen Blocks an.

Jeder Miner im Netzwerk hat somit abhängig von seiner Rechenleistung die Chance einen neuen Block als Erster zu generieren. Hieraus stellt sich die Frage, welche Nutzen der Miner

davon hat seine Rechenleistung für das Netzwerk zur Verfügung zu stellen. Die Antwort ergibt sich aus der Belohnung für das Erstellen von Blöcken. Von der Software werden pro erstelltem Block eine bestimmte Anzahl an Währungseinheiten neu ausgeschüttet. Diese Transaktion, die standardmäßig immer die erste Transaktion eines neuen Blocks ist, beinhaltet neben den neu ausgeschütteten Währungseinheiten auch alle Transaktionsgebühren der Transaktionen, die im erstellten Block vorhanden sind. Daher kommt auch die Motivation Transaktionen mit einer höheren Transaktionsgebühr bevorzug zu behandeln. Da in jedem Durchgang nur ein Block der erste valide Block sein kann, ist hier das Angriffspotenzial groß, da jeder Miner die entsprechenden Währungseinheiten für sich beanspruchen möchte (Bitcoinwiki-G). Eine detaillierte Ausführung möglicher Angriffsszenarien auf die Blockchain werden in Kapitel 4.3 näher betrachtet.

3.4 Kritische Reflexion

Die Blockchain bietet die Möglichkeit ein globales, digitales Zahlungssystem aufzubauen, welches nicht auf Vertrauen in eine dritte Instanz beruht. Durch den Einsatz von digitalen Signaturen kann nur der Eigentümer der Währungseinheiten diese auch ausgeben. Das Problem der Doppelausgabe wird durch die Verknüpfung der Transaktionen und die öffentliche Blockchain, die durch das gesamte Netzwerk validiert wird, gelöst. Die Blockchain kann bei einer entsprechenden Nutzeranzahl und Rechenkapazität des Netzwerks nicht mehr durch eine einzelne Partei geändert werden. Jeder Nutzer kann dem Netzwerk problemlos beitreten und es wieder verlassen. Somit stellt sich also die Frage, welche Einschränkungen bringt diese Technologie mit sich?

Zum einen gibt es **technische Begrenzungen** der aktuellen Blockchain-Technologie wie sie z.B. im Bitcoin eingesetzt wird. Hierbei ist das aktuelle Maximum an Transaktionen die pro Sekunde bearbeitet werden können, auf 7 begrenzt. Zum Vergleich verarbeitet VISA aktuell ca. 2000 Transaktionen pro Sekunde (Swan, 2015, S. 83). Eine weitere technische Hürde stellt die aktuelle Blockgenerierungsrate von einem Block pro 10 Minuten dar. Hierdurch kann es sein, dass der Nutzer 10 Minuten warten muss bis seine Transaktion bestätigt wird. Um sicher zu gehen, dass die Transaktion unveränderbar in der Blockchain gespeichert wurde, sollte der Nutzer 3-4 Blöcke abwarten, was eine Wartezeit von ca. 40 Minuten zur Folge hätte. Des Weiteren muss sich jeder Teilnehmer am Netzwerk eine lokale Kopie der Blockchain laden. Die Blockchain hat aktuell eine Größe von ca. 54 GB und wächst Pro Jahr um ca. 14 GB (Blockchaincenter, 2016). Um eine Wallet bei sich zu installieren, muss somit eine enorme Datenmenge initial geladen werden.

Neben den technischen Restriktionen ist auch die **Sicherheit** ein entscheidender Faktor. Es gibt Angriffsszenarien (wie z.B. 51 %-Attacke) die es möglich machen, dass Änderungen an der Blockchain vorgenommen werden. Des Weiteren ist es auch möglich, dass aktuell genutzte

Kryptografieverfahren geknackt werden und diese somit keine Sicherheit mehr bieten. Daneben ist die Berechnung des Proof-of-Work ein enormer Rechenaufwand, der keinen sinnvollen Nutzen außer dem Erhalt der Sicherheit im Netzwerk hat (O'Dwyer & Malone, 2014). Zuletzt soll auch die Anonymität des Nutzers nicht außer Acht gelassen werden. Durch die öffentliche Blockchain kann theoretische jede Zahlung eingesehen und zurückverfolgt werden. Sobald zu einem öffentlichen Schlüssel die dahinterstehende Person bekannt ist, können alle Zahlungseingänge und -ausgänge ermittelt werden. Der Nutzer kann dieser Problematik durch das Verwenden neuer Schlüssel für jede Zahlung entgegenwirken (Swan, 2015, S. 83 - 87). Das Thema Sicherheit der Blockchain wird in Kapitel 4.3 näher betrachtet.

Neben der Verwendung als Basis für ein digitales Zahlungsmittel kann die Blockchain für diverse weitere Anwendungsszenarien eingesetzt werden. Ein Beispiel hierfür sind Smart Contracts, die ab Kapitel 2.2 näher betrachtet wurden.

4 Prototypische Implementierung einer Blockchain

Definition und grundlegende Funktionsweisen der Blockchain-Technologie wurden im Kapitel 3 dargestellt und näher erläutert. Darauf aufbauend beschäftigt sich das folgende Kapitel mit den Herausforderungen einer Blockchain-Implementation. Zudem erfolgt auf der einen Seite eine Bewertung der allgemeinen Blockchain-Sicherheit, wobei auch alternative Proof-of-Konzepte vorgestellt werden. Auf der anderen Seite wird ein Angriff auf den fertigen Prototyp demonstriert.

4.1 Funktionale und nicht-funktionale Anforderungen an die Implementation

Der Prototyp soll primär der Darstellung der Herausforderungen einer Blockchain-Implementation dienen. Daher zielen seine funktionalen Anforderungen auf den Aufbau einer rudimentären Blockchain. Seine nicht-funktionalen Anforderungen wiederum auf Test- und Nutzungsmöglichkeiten eben dieser.

Funktionale Anforderungen

- Ein Miner muss in der Lage sein eine Blockchain zu erstellen und weiter zu führen.
- Er muss Transaktionen und fremde Blöcke entgegennehmen und verarbeiten.
- Er muss fertige Blöcke seinerseits an das Netzwerk verteilen.
- Er muss inkonsistente/manipulierte Daten innerhalb der Blockchain erkennen.
- Er muss sich im Netzwerk bekannt machen können.

Nicht funktionale Anforderungen

- Ein Nutzer soll die Möglichkeit haben Transaktionen zu senden.
- Ein Nutzer soll Kontostände/die auf ihn gerichteten Transaktionen einsehen können.

4.2 Bestandteile einer Blockchain-Implementation

Die Sichtweise auf die Blockchain wandert im Rahmen eines Entwurfs von der erstmaligen Analyse der einzelnen Bestandteile weg, hin zur Planung einer dezentralen Applikation und der damit verbundenen Kommunikation und Synchronisation der gemeinsamen und gleichsam redundant verteilten Blockkette. Im Folgenden werden die Entwicklungs- und Testumgebung sowie Bestandteile und Funktionen des Prototyps vorgestellt.

4.2.1 Entwicklungs- und Testumgebung

Die Entwicklungsumgebung samt der genutzten Java-Bibliotheken ist gemäß Anhang F Tabelle 5 aufgebaut. Als Testumgebung dienen verschiedene teils virtuelle Computer, wobei auf allen Testrechnern Microsoft Windows7, Windows8 oder Windows10 installiert ist. Als Netzwerk dient ein Heimnetzwerk unter Verwendung einer Fritzbox 7490.

4.2.2 Programmaufbau

Der grundlegende Aufbau sieht gemäß Anhang G Abbildung 34 fünf Bereiche vor:

1. Eine lokale Persistenz Schicht, in der die Blockchain und die Konfigurationsparameter im JSON-Format geschrieben/ausgelesen werden. Als Konfigurationsparameter werden eigene Adressinformationen und auch die anderer Miner hinterlegt.
2. Eine Blockgenerierungseinheit, die kontinuierlich zwischengespeicherte Transaktionen in Blöcke einarbeitet und diese mittels Proof-of-Work finalisiert.
3. Eine Handshake-Einheit, die für die Bekanntmachung im Netzwerk zuständig ist.
4. Eine Datenversendungseinheit, die fertige Blöcke an andere Teilnehmer leitet und ggf. auf „Rückfragen“ antwortet.
5. Eine Datenempfängereinheit, die Transaktionen und Blöcke anderer Miner entgegennimmt und auswertet.

Eine Steuerungsklasse initialisiert alle fünf Bereiche und stellt während des Programmablaufs allen Klassen Konfigurations- und Objektparameter zur Verfügung.

4.2.3 Grundelemente

Die Basiselemente der prototypischen Blockchain entsprechen grundlegend dem Aufbau der Bitcoin Blockchain-Architektur (Siehe Kapitel 3.1) und sind auf Abbildung 11 zu sehen.

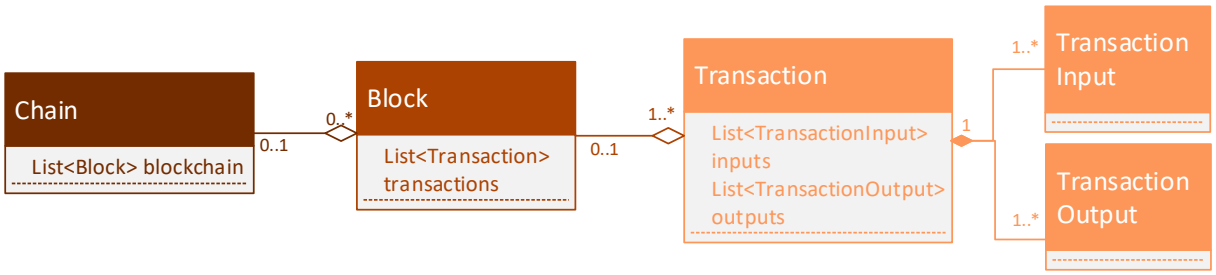

Abbildung 11: Grundelemente eine Blockchain

Eine Kette (Chain) enthält eine Liste von Blöcken (Block), ein Block wiederum eine Liste von Transaktionen (Transaction) und jede Transaktion jeweils eine Liste für Transaktionseingänge (Transaction Input) und -ausgänge (Transaction Output).

Transaktionen können mehrere Eingänge haben, da der gesamte ausgehende Betrag mit der Wertigkeit kumulierter Eingänge bedient werden muss.

Da Transaktionseingänge nicht geteilt, sondern einmalig verwendet werden und deshalb Transaktionsrestwerte (Summe aller Eingänge – Ausgangsbetrag) entstehen, muss auch die Möglichkeit eines weiteren Transaktionsausgangs bestehen, mit welchem der entstehende Restbetrag an den Sender übertragen werden kann.

Anders als in der Bitcoin Blockchain werden Transaktionen und Blöcke im Prototyp durchgehend nummeriert, anstelle dessen sie anhand von Zeitstempeln zu verbinden. (vgl. Kapitel 3.3.3) Diese Nummerierung erlaubt eine einfache Sortierung, unabhängig davon in welcher Form die daraus bestehende Blockchain persistiert wird.

Ein weiterer Unterschied liegt in der Behandlung der Transaktionsausgänge, welche im Gegensatz zu der Bitcoin Blockchain nicht durchnummeriert werden. Die Ausgänge werden im Prototyp so durch ihren Transaktions-Hash und ihre Zieladresse identifiziert, ohne ihre Position innerhalb der Transaktionsausgänge zu berücksichtigen.

4.2.4 Blockgenerierung

Die Blockgenerierung ist generell durch das angewandte Proof-of-Work-Verfahren (vgl. Kapitel 4.2.6) mit einem hohen Zeitaufwand verbunden, auch wenn dieser im Rahmen des Prototyps nur einige Sekunden einnimmt. Dies schafft den Anreiz stets einen Block zu generieren und parallel Transaktionen für den nächsten Block zu sammeln.

Eingehende Transaktionen (vgl. Kapitel 4.2.7) werden in eine Warteschlange aufgenommen, um eine asynchrone Verarbeitung zu ermöglichen. Im Rahmen der Blockgenerierung werden die Transaktionen auf Korrektheit geprüft und in einen neuen Block gegossen.

Die Transaktionsprüfung besteht dabei aus mehreren Teilprüfungen. Zunächst wird die Konsistenz geprüft, indem die Gleichheit der kumulierten Eingangs- und Ausgangswerte sichergestellt wird. Darauffolgend wird in der Blockchain für jeden Transaktionseingang nach einer existierenden Quell-Transaktion gesucht, in welcher der Transaktions-Hash den angegebenen Eingangs-Hash entspricht und die gleichsam über einen Ausgang mit passender Zieladresse und Wertigkeit verfügt. Wenn diese gefunden wurde wird zuletzt nach einer Transaktion gesucht, welche genau diese bereits als Eingang nutzt, um Doppelausgaben auszuschließen. Wenn die Konsistenz gegeben ist, alle Transaktionseingänge existieren und gleichsam noch nicht benutzt wurden, gilt die Transaktion als gültig und wird in den Block aufgenommen.

Zusätzlich wird jeder Block mit einer speziellen Belohnungstransaktion ausgestattet, welche dem Miner, analog zur Bitcoin Blockchain, einen Anreiz geben soll, Prozessorzeit zu investieren. Ferner stellt diese Belohnung die einzige Art der digitalen Coin-Generierung dar, wodurch sie essenziell für jede Kryptowährung ist. Anders als in der Bitcoin Blockchain werden Transaktionen dabei zwecks Komplexitätsminderung nicht mit einer Transaktionsgebühr versehen.

Schlussendlich folgt die Finalisierung des Blocks, indem zunächst der Transaktions-Hash und schließlich der Blockhash mittels Proof-of-Work ermittelt wird. Der fertige Block wird dann

der lokalen Chain beigefügt und der Sendeinheit übergeben, die ihn an alle bekannten Netzwerkteilnehmer verteilt.

Da nicht nur Transaktionen, sondern auch Blöcke eingehen (vgl. 4.2.7) muss der Prozess der Blockgenerierung auch abgebrochen werden können, falls ein Block eingeht, welcher die gegenwärtige Blockchain weiter führt. Dabei muss berücksichtigt werden, dass der Abbruch nicht zu Instabilität oder zum Verlust von eingegangenen Transaktionen führen darf. Diese Problematik löst der Prototyp mit den folgenden zwei Methoden:

1. Nicht nur die allgemeine Blockgenerierungseinheit, welche innerhalb einer Endlosschleife Generierungsaufträge initiiert, sondern auch die einzelnen Generierungsaufträge erfolgen in separaten Threads. Diese werden bei Bedarf mittels InterruptedException (*thread.interrupt()*) kontrolliert unterbrochen, indem die Exceptions in den betreffenden Threads abgefangen und behandelt werden.
2. Anstelle dessen Transaktionen direkt aus der Queue zu entnehmen und sie im Falle eines Abbruchs und Verwurfs des sich in Erstellung befindenden Blocks zu verwerfen, wird zunächst ein Klon der Queue angefordert. Nach erfolgreicher Erstellung und Einbindung des neuen Blocks in die Chain können dann die verarbeiteten Transaktionen aus der aktuellen Queue entfernt werden.

Eine detaillierte Modellierung des Ablaufs der Blockerstellung ist unter Anhang H in Abbildung 38 und Abbildung 39 zu sehen.

4.2.5 Kryptografische Prüfungen

Grundsätzlich muss eine Kryptowährung in Rahmen der Blockerstellung, der Blockannahme und der Annahme von Transaktionen kryptografische Prüfungen durchführen. Dabei ist es erforderlich zwischen Authentizitätsprüfungen mit Hilfe digitaler Signaturen innerhalb der Annahme von Transaktionen auf der einen, und Integritätsprüfungen mit Hilfe von Hashfunktionen im Rahmen der Blockerstellung und -annahme auf der anderen Seite zu unterscheiden.

So muss in der Praxis innerhalb einer jeden Blockchain sichergestellt werden, dass jede Transaktion von einem eindeutig identifizierten Teilnehmer autorisiert wurde, was mittels asymmetrischer Verschlüsselungsverfahren erreicht werden kann. (vgl. Kapitel 3.2.1) Im Rahmen eines Prototyps, der die wesentlichen Merkmale einer Blockchain inne haben und lediglich für Testzwecke zur Verfügung stehen soll, kann hingegen zunächst auf diese Prüfung verzichtet werden. Somit wird diese Prüfung innerhalb dieser Arbeit nicht weiter behandelt und jede eingehende Transaktion als autorisiert eingestuft.

Wesentlicher und unverzichtbarer Bestandteil einer Blockchain hingegen ist die Integritätsprüfung, da die Blockchain als Kette von jeweils mit dem Vorgänger-Hash versehen Blöcken bereits grundsätzlich auf Hashfunktionen basiert. Der Einsatz von Integritätsprüfung und Hashfunktionen und deren Funktion im Prototyp werden im Folgenden unter Bezug auf das Klassendiagramm (Anhang G Abbildung 35) beschrieben.

Eingesetzte Hash-Funktion

Für den Prototyp wird die Java-eigene hashCode()-Funktion der Klasse String genutzt. Die Berechnung entspricht dabei der Formel: $h(s) = \sum_{i=0}^{n-1} s[i] \times 31^{n-1-i}$. (Oracle, 2010) Somit müssen alle zu hashenden Werte zum Zeitpunkt des Funktionsaufrufs vom Typ String sein. Auf den Einsatz von SHA-Algorithmen (vgl. Kapitel 3.2.2) wird zunächst bewusst verzichtet, um die Komplexität des Prototyps möglichst gering zu halten.

Unterschiede zur Hash-Funktionen der Bitcoin Blockchain

Im Gegensatz zur Bitcoin Blockchain werden weder Hashwerte von Transaktionseingängen und -ausgängen, noch die Hashwerte der Transaktionen eines Blocks baumartig und Paarweise rekursiv gehasht, bis nur noch ein Transaktionshash besteht (vgl. Kapitel 3.1.2), sondern lediglich aufaddiert. Dies verringert die Sicherheit, verschlankt aber gleichsam den Code und wird für Testzwecke als ausreichend erachtet. Zudem werden alle Hash-Funktionen nur einmal und nicht mehrfach durchlaufen.

Einsatz in Transaktionen

Jede Transaktion hat eine Reihe von Inputs und gleichsam ein bis zwei Ausgänge[1], wobei jedes Objekt des Typs *Transaction* (Anhang J Transaction.java), *Input* (TransactionInput.java) oder *Output* (TransactionOutput.java) über eine eigene *hashCode()* Methode verfügt. Die *hashCode*-Funktion des Objekts *TransactionInput* lautet $h(iH, iV) = (31 + iH) * 31 + iV$ [2] und die des Objekts *TransactionOutput* $h(oT, oV) = (31 + oT) * 31 + oV$ [3]. Der Hash einer Transaktion ergibt sich aus dem Hash der Vorgänger Transaktion, der laufenden Transaktionsnummer, der Senderadresse, der Transaktionseingänge und der Transaktionsausgänge und wird dabei wie folgt berechnet: $h(S, V, T, TE, TA) = S + V + T + \sum hashCode(TE) + \sum hashCode(TA)$ [4].

---

[1] Adressen des Transaktionsempfängers und des Transaktionssenders für evtl. Restwerte vermerkt. Die Summe der Inputs muss gleich die der Wertigkeit der kumulierten Outputs entsprechen.

[2] iH = Eingangs-hash, iV = Eingangswert.

[3] oT = Ausgangsziel, oV = Ausgangswert.

[4] S = Senderadresse, V = Vorgängerhash.

T = Transaktionsnummer, TE = Transaktionseingang, TA = Transaktionsausgang.

Einsatz in Blöcken

Der Hash eines Blocks ergibt sich aus der Berechnung des Proof-of-Work. Zur Durchführung werden die *Challenge* und die PoW-Schwierigkeit benötigt, wobei sich die *Challenge* aus der Summe dieser Werte ergibt: Der Hashwert des vorangegangenen Blocks, die laufende Blocknummer und der Transaktions-Hash. Der Transaktionshash wird dafür aus den als String aneinander geketteten Hashwerten der Einzelnen Transaktionen gebildet. ($h(t) = String.hashCode(String(t_1.hashcode() + \cdots + t_n.hashcode()))$, Anhang J Block.java – getTransactionHash() : int)

Die Funktionsweise des PoW sowie der Gebrauch der PoW-Schwierigkeit und der *Challenge* werden im Kapitel 4.2.6 näher betrachtet.

Einsatz in der Blockkette

Die Blockkette selbst verfügt über keine Hash-Methode, sondern über Funktionen die der Überprüfung der gesamten Chain dienen. Hier wird zum Programmstart die Integrität der gesamten Blockchain überprüft. (Anhang J Chain.java – getLastValidBlock() : int)

Einsatz in der Netzwerkkommunikation

Jeder aus dem Netzwerk eintreffende Block muss hinsichtlich seiner Integrität geprüft werden, bevor er in die eigene Blockchain integriert werden kann. Innerhalb einer geschlossenen Testumgebung wird jedoch zunächst auf diese Prüfung verzichtet, da jeder Teilnehmer über die identische Anwendung als Miner der Blockchain verfügt, wodurch jedwede unkontrollierten Angriffsversuche ausgeschlossen sind.

4.2.6 Proof-of-Work

Im Rahmen der Blockfinalisierung werden zunächst der bereits zuvor ermittelte Transaktionshash, der Hash des vorangegangenen Blocks und die Blocknummer in einer *Challenge* als Integer Variable aufaddiert.

```
private int proofOfWork(int challenge,
  int n) throws NoValidPoWException {
  int blockHash;
  for (int pow = -2147483647;
    pow <= 2147483647; pow++){
    blockHash = ((Object)
      (challenge * pow)).hashCode();
      if (startsWithNZeros(blockHash, n)){
        this.blockhash =
          Integer.valueOf(blockHash);
        return pow;
      }
    }
  throw new NoValidPoWException();
}
```

Abbildung 12: Codeausschnitt: Proof-of-Work

Innerhalb der Methode *proofOfWork* (Abbildung 12) wird daraufhin iterativ ein Wert gesucht, welcher multipliziert mit der *Challenge* ein Produkt ergibt, aus dem sich ein Hashwert mit *n* führenden Nullstellen ergibt (entspricht der Schwierigkeit). Dieser Wert ist der Proof-of-Work. Der sich aus dem Produkt ergebene Hash ist der Blockhash.

4.2.7 Netzwerkkommunikation

Die Netzwerkkommunikation unter Minern besteht aus zwei Bereichen: Zum einen aus der Verbindung und Bekanntmachung, zum anderen aus dem Austausch von Transaktionen und Blöcken.

Die Bekanntmachung und somit auch der Zugang zum Mining-Netzwerk erfolgt gemäß Abbildung 13, wobei von einem geschlossenen LAN ausgegangen wird. Jede Node sendet mittels *TimerTask* in definierten Zeitintervallen ein *DatagramPacket* in Form eines Broadcasts in das Netzwerk, wobei sich innerhalb des *DatagramPackets* ein JSON-Objekt mit den Adressdaten der jeweiligen Node und der Typisierung „handshake“ befindet. (Anhang J AddressDataSender.java) Gleichsam hält jede Node einen *DatagramSocket* aufrecht, welcher eingehende Broadcasts annimmt und anhand des angegebenen Typs interpretiert. (Anhang J AddressDataReceiver.java) Eingehende JSON-Objekte des Typs „handshake“ werden akzeptiert und die sich darin befindenden Adressinformationen im Falle bisheriger Unbekanntheit in die Liste der bekannten Hosts aufgenommen. Zudem werden einem bisher ungekannten Broadcast-Sender im Gegenzug gezielt die Node-eigenen Adressinformationen gesendet. (Anhang J Controller.java – addHost(Host))

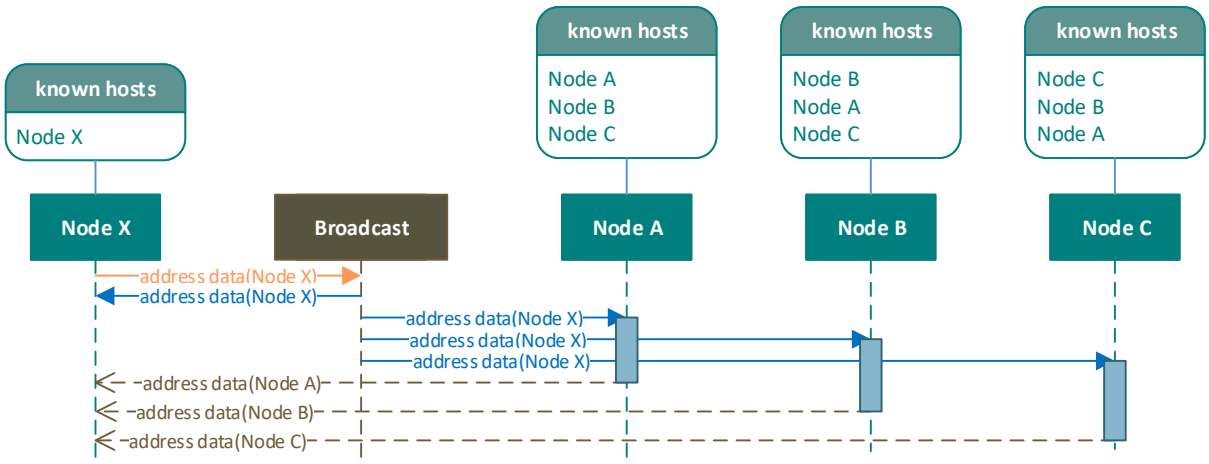

Abbildung 13: Broadcast der Mineradresse

Die Konzeption der Nodes im Bereich der Transaktions- und Blockaustausch-Kommunikation erfolgt gemäß folgender Definition der Bitcoin-Implementation: „The Bitcoin network is a network of homogeneous nodes. There are no coordinating roles and each node keeps a

complete replica of all the information needed to verify the validity of incoming transactions. Each node verifies information it receives from other nodes independently and there is only minimal trust between the nodes. (Decker & Wattenhofer, Information Propagation in the Bitcoin Network, 2013, S. 3)“ Innerhalb der Implementation wird jedoch zwecks Vereinfachung der Block Generierung anderer Nodes Vertrauen geschenkt. (Vgl. Kapitel 4.2.5) Für den Blockaustausch bedarf es zweier Eigenschaften: Die des Senders (Anhang J NetworkSender.java) und die des Empfängers (Anhang J NetworkReceiver.java). Diese werden im Folgenden beschrieben.

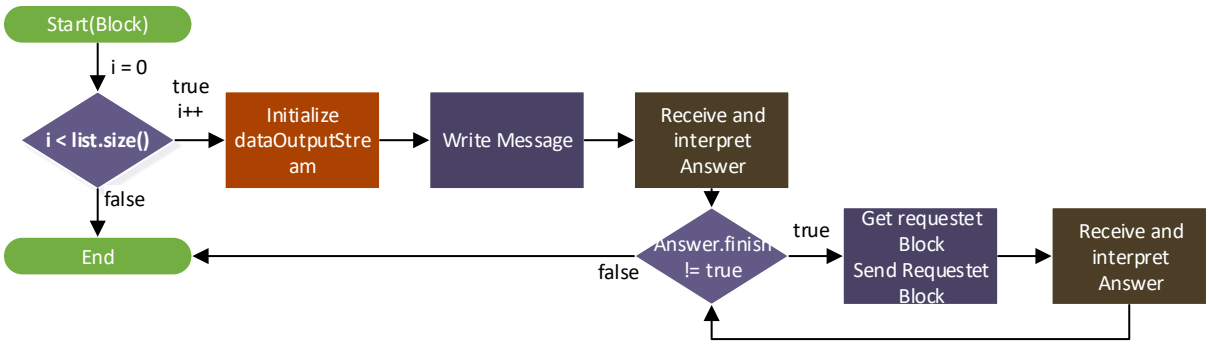

Abbildung 14: Versendung eines Blocks

Die Versendung eines fertigen Blocks verhält sich entsprechend Abbildung 14. Dabei wird der einzelne Versendungsprozess für jeden Miner in der Liste der bekannten Hosts durchgeführt. Zunächst wird der fertige Block als JSON-Objekt in ein Byte Array geladen, dann der Socket entsprechend des Ziel Hosts initialisiert und schließlich ein *Dataoutputstream* auf Basis des Sockets erstellt, mit dem das Byte Array übertragen werden kann.

Nach der erfolgreichen Blockübertragung wird eine Antwort (ebenfalls im JSON-Format) erwartet, welche mittels *BufferedReader* entgegengenommen wird. In dieser kann der Empfänger bei Bedarf weitere Blöcke anfordern, indem er das Attribut *finish* mit *false* belegt und die Nummer des nächsten zu sendenden Blocks übergibt. Sobald der Empfänger alle notwendigen Blöcke erhalten hat antwortet er mit einem JSON-Objekt, das mit dem Wertepaar *finish* und *true* belegt ist. Dadurch ist der Versendungsprozess abgeschlossen.

Im Rahmen des Empfängers wird der Empfang von Transaktionen und Blöcken unterschieden. Transaktionen werden direkt und ohne Prüfung in die Warteschlange der Empfänger-Node übernommen, da eine Prüfung innerhalb der Blockerstellung ausreichend ist. (Vgl. Kapitel 4.2.4) Empfangene Blöcke hingegen durchlaufen einen Entscheidungsbaum, da sie nach den Regeln der Blockchain abgehandelt werden müssen. Dieser Entscheidungsbaum ist auf Abbildung 15 zu sehen und wird im Folgenden erläutert.

Für den korrekten Umgang mit einem eingehenden Block muss zunächst der letzte Block der aktuellen Blockchain *BlockN* herangezogen werden. Dabei gilt es nur die eingehenden Blöcke

näher zu betrachten, deren Blocknummer größer gleich der des *BlockN* ist. Dem Prototyp liegt dazu die Annahme zugrunde, dass stets die längste Blockchain zu bevorzugen ist.[5] Neben der zu ermittelnden Blocknummer sind auch stets die Block Hashwerte und die mitgeführten PreHashes (Hash des vorhergehenden Blocks) von Bedeutung, da so geprüft werden kann, ob ein eingehender Block zur bisherigen Chain passt. Zudem ist auch die Möglichkeit alternativ geführter Blockchain-Branches gegeben, in welcher eingehende Blöcke geführt werden, die gleichauf mit der aktuellen Chain sind.

Anhand der auf Abbildung 15 zu sehenden Verzweigungen kann der Block so direkt zur bestehenden Chain hinzugefügt oder abgewiesen werden, oder weiteres Handeln erfordern. Dies bedeutet im Falle eines empfangenen Blocks, welcher der geführten Blockchain weit voraus ist, alle Blöcke bis zur Blocknummer von *BlockN* + 1 vom Sender zu erfragen. Im Falle eines weiterführenden Blocks oder einer weiterführenden Blockreihe, in welcher der direkte numerische Nachfolger von *BlockN* nicht über den passenden *PreHash* verfügt, muss der Receiver prüfen, ob es eine passende parallel geführte Chain gibt. Wenn dem so ist, muss fortan diese als primäre Blockkette geführt werden und der Block/die Blockreihe kann ergänzt werden. Andernfalls ist ein *Merge* erforderlich, wobei der aktuell letzte eigene Block solange gegen einen des Senders ausgetauscht wird, bis dessen PreHash dem Hash seines numerischen Vorgängers entspricht. Zum Schluss einer jeden Blockentgegennahme werden schließlich die alternativen Blockketten, welche kürzer als die nun bestehende Kette sind, verworfen.

[5] Hingegen gilt innerhalb der Bitcoin Blockchain das Gesetz der höchsten Komplexität (vgl. Kapitel 3.1.3): Kumulierte Komplexität der PoWs > Blockchain Länge.

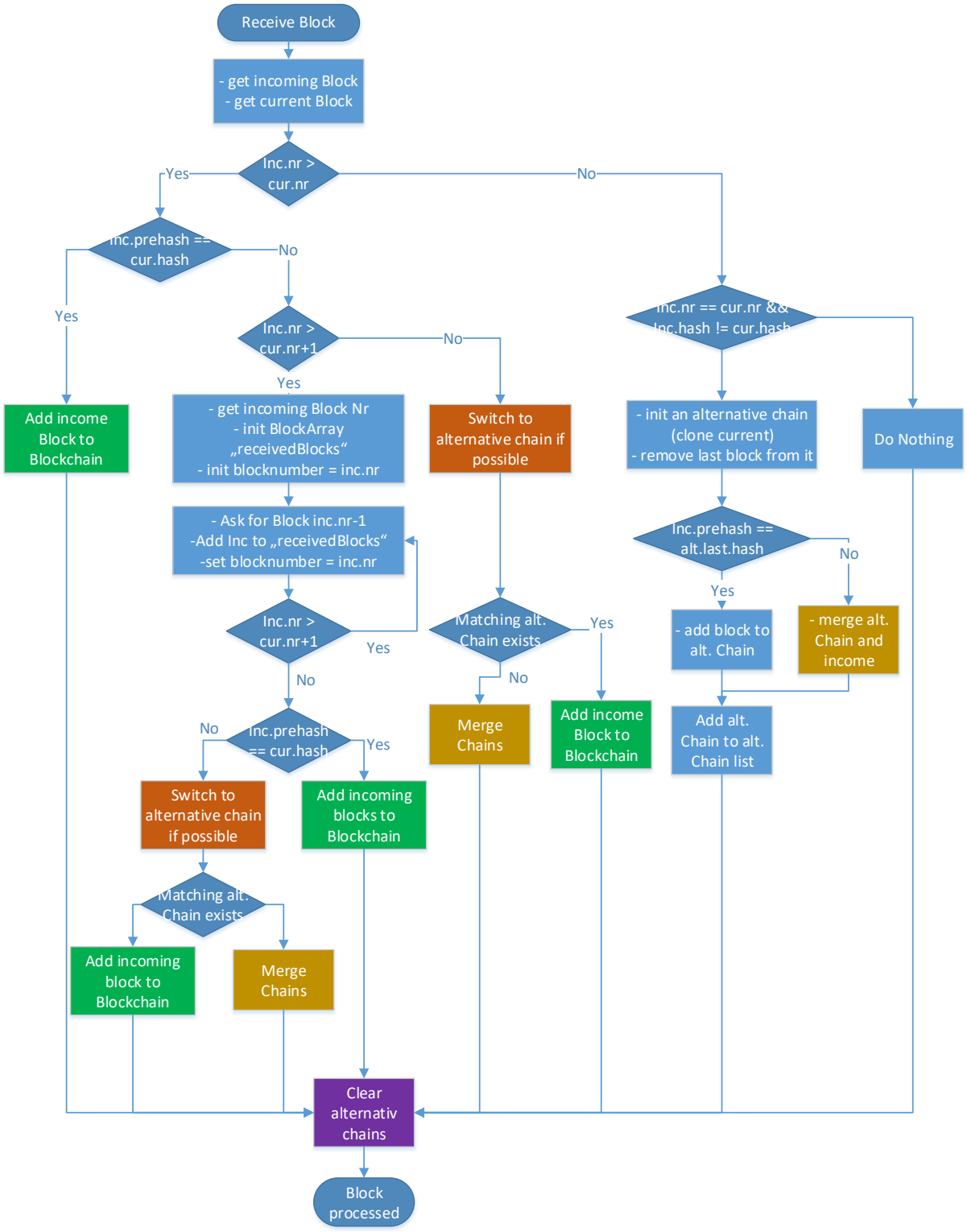

Abbildung 15: Empfang eines Blocks

4.2.8 Wallet-Anwendung

Für die Nutzung der Blockchain seitens eines Anwenders ist eine Wallet-Anwendung erforderlich. Die Mining-Anwendung muss dafür zunächst um die Übermittlungsmöglichkeit der ganzen Blockchain als JSON ergänzt werden. Gemäß Anhang G Abbildung 36 wurde dann

eine Anwendung konzipiert, mit der ein Konto innerhalb der Blockchain genutzt werden kann, ohne dass der Nutzer die zugrundeliegende Technologie kennen muss. So fordert die Wallet-Software die gesamte Blockchain eines Miners an, um diese auf noch nicht benutzte Transaktionen, die auf den Nutzer gerichtet sind, zu untersuchen. Die Summe dieser Transaktionen bildet den angezeigten Kontostand. Für ausgehende Transaktionen bündelt die Anwendung im Gegenzug Eingänge so zusammen, dass deren Summe den erforderlichen Ausgangsbetrag bildet. Eventuelle Restwerte werden in Form eines weiteren Outputs zurück an den Nutzer selbst geschickt.

4.3 Sicherheit einer Blockchain – Angriffsszenarien

Wenn Sicherheitsmechanismen im Rahmen der Anwendung von Blockchain Technologien betrachtet werden, muss zunächst zwischen der Sicherheit der Anwendungen (z.B. Wallet Anwendungen) und der Sicherheit der zugrundeliegenden Technologie, also der Blockchain, unterschieden werden. Gezielte Angriffe (Kannenberg, 2015) oder auch Nutzung von groben Sicherheitslücken (Eikenberg, 2015) im Anwendungsbereich erwecken den Anschein einer angreifbaren Technologie, beruhen dabei jedoch lediglich auf unausgereiften Entwicklungen von Wallet-Anbietern und sollen im Folgenden nicht näher betrachtet werden. Auch die Sicherheit der kryptografischen Methoden und Funktionen, welche die grundlegende Basis einer jeden Blockchain darstellen (Kapitel 3.2), wird im Folgenden als gegeben hingenommen, um somit die Sicherheit der Blockchain und der mit ihr verbundenen Neuerungen zu betrachten.

Mit dem Proof-of-Work wird die Gesamtrechenleistung des Netzwerks die grundlegende Basis einer sicheren Blockchain. Eben dieses Verfahren, welches unter anderem beim Bitcoin zum Einsatz kommt, ist jedoch mit immensem Rechenaufwand und damit einhergehenden Kosten verbunden. Je nach eingesetzter Hardware bedeutet dies, „that the total power used for Bitcoin mining is around 0.1-10 GW. (O'Dwyer & Malone, 2014)" Der durchschnittliche elektrische Energieverbrauch Irlands wird auf ca. 3 GW beziffert (Eurostat, 2016), ist also durchaus vergleichbar mit dem des Bitcoin-Netzwerks. Die daraus resultierende Kritik hat zu Weiterentwicklungen des PoW und gänzlichen Neuentwicklungen alternativer Sicherungsmechanismen beigetragen, welche im Folgenden vorgestellt werden sollen.

4.3.1 Alternative Proof-Of-Konzepte

Proof-of-Stake

Die Stimmgewalt eines Miners im Proof-of-Work folgt aus seiner Rechenleistung, da diese die Wahrscheinlichkeit der Blockgenerierung im Netzwerk und somit auch seinen Einfluss bestimmt. Die grundlegende Idee des Proof-of-Stake (PoS) ist die Entkoppelung der

Stimmgewalt eines Miners von seiner Rechenleistung, hin zur Gewichtung anhand der Anzahl der sich in seinem Besitz befindenden Coins. Der durch den PoW entstehende Rechenaufwand und die damit verbundenen Ressourcen sollen somit eingespart werden.

In der Anwendung bedeutet dies, dass mit jedem Tag des Besitzens eines Coins eine Art Stimmberechtigung wächst, wobei je nach Konvention eine Mindest- und Höchstdauer bestimmt wird, die ein Coin im Besitz sein muss um die Wahrscheinlichkeit den nächsten Block generieren zu dürfen zu erhöhen. Bei Erfolg wird diese Gewinnchance wieder auf null gesetzt wodurch die zeitliche Messung des Besitzes von Neuem startet.

Die bei der Blockgenerierung als Belohnung geschaffenen Coins und einbehaltene Transaktionsgebühren werden zufällig und periodisch über die Coin-Besitzer ausgeschüttet, wobei die Ausschüttungswahrscheinlichkeit eines Teilnehmers anhand der kumulierten Wertigkeit der von ihm repräsentierten Coins und der Dauer der Verfügungsgewalt über eben diese ermittelt wird. (QuantumMechanic, 2011)

Als Schutz vor Manipulation wird nun anders als beim PoW nicht die Rechenleistung, sondern das Vermögen herangezogen. Ein Angreifer muss über mehr als die Hälfte des Gesamtvermögens verfügen, um über eine ausreichend hohe Wahrscheinlichkeit zu verfügen eine gültige parallele Chain führen zu können. Neben Kryptowährungen wie NXT (Nxt Community, 2015), welche gänzlich auf PoS setzen, ist auch der Einsatz von hybriden Proof-of-Verfahren beliebt und wird in Kryptowährungen wie Peercoin (Peercoin, 2015) genutzt. Hier werden verschiedene Kombination und Anteilsmischungen aus PoW und PoS genutzt, um beide Vorteile zu nutzen und Nachteile zu minimieren.

Proof-of-Activity

Mit dem Proof-of-Activity wird die Idee einer hybriden Blockchain weiter verfolgt, indem zur gleichen Zeit PoW- und PoS-Blöcke gesucht werden. Der Idee ist bisher noch keine Umsetzung gefolgt, allerdings bietet die angestrebte Art der Block Generierung Vorteile hingegen einer reinen PoW- oder PoS-Methodik. (Patterson, 2015)

So wird zunächst ein PoW mit vorgegebener Komplexität gesucht und nach erfolgreicher Suche ins Netzwerk gesendet. Anders als beispielsweise im Bitcoin-Netzwerk handelt es sich hier aber noch nicht um einen fertigen Block, denn zunächst wird der erzeugte Block Hash in N Nummern aufgeteilt, wobei jede Nummer auf den Public Key eines beliebigen Coin-Besitzers verweist. Der Block gilt dann als vollwertig, wenn eine vordefinierte Anzahl der N zufälligen Besitzer den Block mit ihrem jeweiligen Private Key signiert haben. (Bentov, Lee, Mizrahi, & Rosenfeld, 2014)

Die Belohnung für die Block Generierung wird zwischen dem Proof-of-Work-Miner und den signierenden Teilnehmern aufgeteilt. Somit wird zum einen die Arbeit belohnt und zum anderen eine Art Verzinsung des Guthabens gewährleistet.

Die im Verfahren inbegriffene Mehrfachprüfung eines Blocks erschwert somit nochmals eine Manipulation der Blockchain. Weder dem Netzwerk überlegene Rechenleistung noch der Besitz des Großteils allen Umlaufvermögens allein genügen für die realistische Umsetzbarkeit von nachträglichen Veränderungen oder Mehrfachausgaben.

Proof-of-Burn

Die grundsätzliche Idee hinter dem Proof-of-Burn (PoB), basierend auf dem Whitepaper von P4Titan (2014), lautet in Anbetracht der Schwierigkeit der Blockerstellung: „all that matters is that an *individual miner* finds the task expensive." (Bitcoinwiki-N, 2016). Dieser Aussage folgend wird unter Nutzung des Konzepts PoB keine Rechenleistung verbrannt, sondern digitale Coins, also das Vermögen der Miner. Das *Verbrennen* einer Münze erfolgt dabei durch den Versand an eine spezielle im Source Code hinterlegte Adresse.

Grundsätzlich ist das Verfahren dem Proof-of-Stake, das ebenfalls eine Zufallsverteilung der Belohnungen vorsieht, recht ähnlich. So ist eine Unterscheidung anhand der folgenden drei Punkte möglich:

1. Anstelle des kumulierten Coin-Alters (Age) werden die Coins als Ganzes verbrannt.
2. Eine Verbrennungsausgabe erfolgt erfolgsunabhängig, wohingegen das Age nur bei Erfolg zurückgesetzt wird.
3. Die Chancenerhöhung für den Erhalt jeder einzelnen Belohnung ist dauerhaft, wodurch das Verbrennen von digitalen Münzen zur Investition wird.

Der Einsatz von Proof-of-Burn ist, bedingt durch das vorausgesetzte digitale Umlaufvermögen, nur in Verbindung mit einem Mining Verfahren möglich. So muss zunächst Vermögen geschürft werden, um dann verbrannt werden zu können. Ein Beispiel einer solchen Verbindung ist die Kryptowährung XCP (Counism, 2014), welche basierend auf der Bitcoin Blockchain um das Proof-of-Burn erweitert wurde. Ein Angriff auf eine solche Währung ist für den Angreifer sehr riskant, da er mehr als die Hälfte aller verbrannten Coins stellen muss, wodurch sein eingesetztes Vermögen anders als bei Proof-of-Stake basierten Währungen bei Misserfolg unwiederbringlich weg ist.

Proof-of-Capacity

Dem Namen entsprechend setzt der Proof-of-Capacity (PoC) anstelle von Rechenleistung auf Speichergröße und wird bereits in der Kryptowährung *Burstcoin* (Burst, 2015) eingesetzt. Neben der angestrebten erhöhten Energieeffizienz soll somit auch der Schutz vor Bot-Netzen gewährleistet werden. Der Aspekt eines Angriffs durch das Leihen von Rechner Ressourcen

via Trojaner wird somit in den Fokus gerückt und mit einer inhärenten Lösung abgewickelt: Der Speicherplatzbedarf durch das Verfahren ist so groß, dass eine Infizierung durch einen terabyte-großen Trojaner schwerlich unbemerkt bleibt. Als Vorteil gegenüber dem Proof-of-Work kann dies aber nur bedingt gewertet werden, da auch eine kontinuierliche vollständige CPU Auslastung durch Hash Operationen den infizierten Rechner verlangsamen wird und der Trojaner somit enttarnt werden kann. (Patterson, 2015)

Das Mining Prinzip des PoC sieht vor, dass Miner Datensegmente (*Plots*) generieren und auf der Festplatte speichern. Mit der Indexnummer des letzten Blocks der aktuellen Chain wird dann aus jedem *Plot* eine *Scoop* (Schaufel) genommen. Schließlich ist analog zum Proof-of-Work ein PoC gefunden, wenn der gemeinsame Hashwert von *Scoop* und dem letzten Block Header, je nach Schwierigkeit, mit *n* Nullstellen beginnt. (Cordell, 2014) Jedes Megabyte allokierter Speicher kann also als zusätzliches Los für die Findung eines PoC betrachtet werden.

Mit einer solchen Implementierung birgt der PoC aber zwei Probleme: Zum einen kann ein Miner mangels Rechenaufwand ohne Kosten gleichzeitig durch Untersuchung aller *Plots* an beliebig vielen alternativen Chains arbeiten. Zum anderen erfordert der Speicherzugriff auf eine Festplatte Zeit (im Schnitt 9ms (ITWissen, 2016)), wodurch die spontane Erstellung von *Scoops* im Hauptspeicher effizienter sein kann. Damit wäre wiederum ein simpler Proof-of-Work Algorithmus implementiert.

Weitere Ideen

Neben den hier genannten Konzepten existieren noch einige weitere, welche zum großen Teil als leicht abgewandelte Kopien des Proof-of-Work oder Proof-of-Stake abgetan werden können. Es gibt hingegen auch andere Varianten, wie der Proof-of-Storage mit dem Entwürfe dezentralisierter Cloud Speicher umgesetzt werden. (Ateniese, Kamara, & Katz, 2009) Solche verhältnismäßig exotischen Verfahren bergen unter Nutzung von Blockchain-Technologie und Netzwerkprotokollen neue und andere Möglichkeiten, sodass sie gänzlich separat betrachtet werden müssen und innerhalb dieser Arbeit nicht weiter untersucht werden können.

4.3.2 Theoretische Angriffe auf Blockchain-Konzepte

Bei Nutzung einer Blockchain-Technologie sind verschiedene Angriffsszenarien denkbar. An dieser Stelle sollen zwei Angriffsszenarien vorgestellt und darauffolgend in Hinblick auf ihren möglichen Erfolg je nach Proof-of-Konzept untersucht werden: Die Doppelausgabe und die nachträgliche Veränderung. Als zu untersuchende Proof-of-Konzepte werden dabei PoW und PoS herangezogen, da sie die höchste praktische Relevanz haben. (Vgl. Kapitel 2.1)

Die Doppelausgabe

Die Grundidee einer Doppelausgabe liegt darin, eine digitale Münze stets kopieren und an mehrere Teilnehmer gegen eine Gegenleistung eintauschen zu können. (Bitcoin.org, 2015a) Dies wird innerhalb der Blockchain verhindert, indem ein Transaktionsausgang nur einmal innerhalb der gesamten Blockchain-Historie verwendet werden kann. (Vgl. 4.2.4)

Um dennoch das Ziel der Täuschung eines Teilnehmers zu erreichen, muss ein Angreifer analog der Abbildung 16 zur gleichen Zeit zwei Blöcke, A und B, erstellen, wobei Block A eine an das Betrugsopfer gerichtete und Block B eine an den Sender selbst gerichtete Transaktion gleichen Ursprungs enthält.

So soll der Betrogene Block A erhalten, bei Prüfung feststellen, dass die empfangene Transaktion gültig ist und eine dafür vereinbarte Gegenleistung erbringen. Gleichsam erhält das Mining-Netzwerk Block B, der die an den Betrüger selbst gerichtete Transaktion enthält, und befindet diesen Block als gültig, wodurch Block A und die sich darin befindende Transaktion nachträglich unwirksam wird.

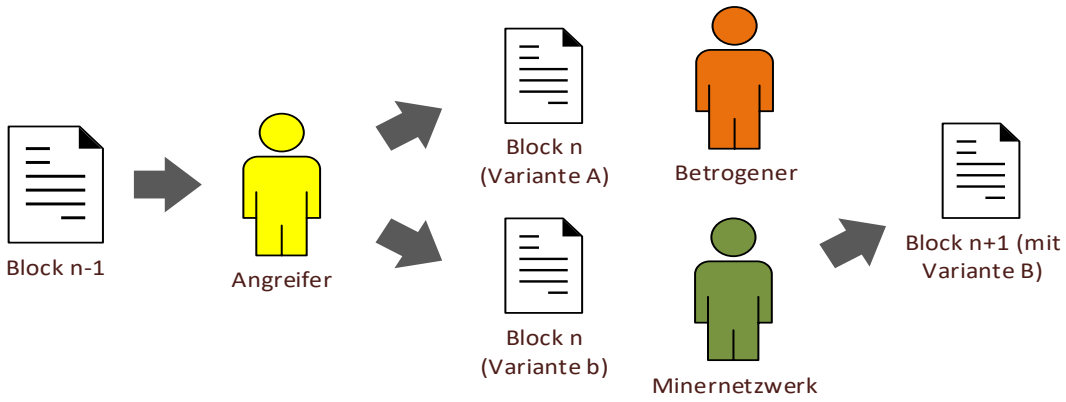

Abbildung 16: Erfolgreiche Doppelausgabe

Die nachträgliche Veränderung

Die Grundidee der nachträglichen Veränderung der Blockchain beruht darauf, dass ein einmal erstellter und allgemein anerkannter Block niemals final und somit endgültig ist. Eine längere bzw. komplexere Kette als die bisher anerkannte kann also zum Austausch aller bestehenden Blöcke führen. (Bitcoin.org, 2015b) Neben destruktiv motivierten Anreizen eines solchen Angriffs bestehen auch finanzielle Reize im Form der Aneignung von Belohnungstransaktionen und der Abänderung bestehender Transaktionen zu Gunsten des Angreifers.

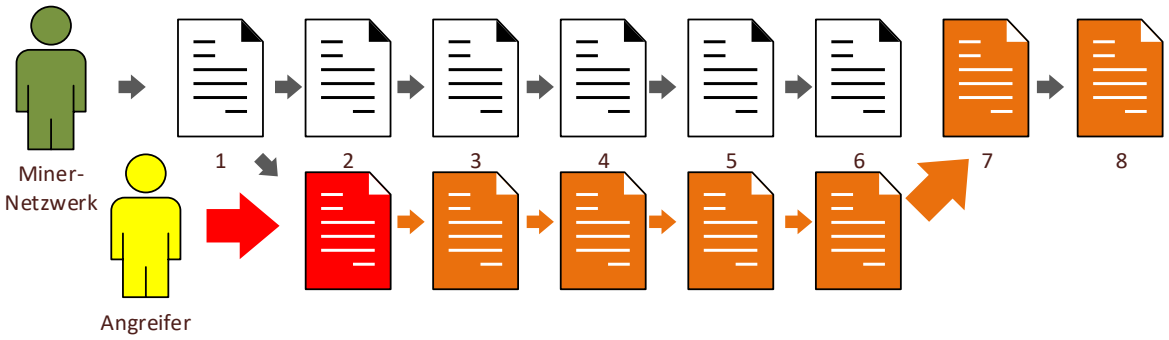

Abbildung 17: Nachträgliche Veränderungen durch einen Angreifer

Die Durchführung eines solchen Angriffs würde entsprechend Abbildung 17 erfolgen, indem das Miner-Netzwerk bereits eine Blockkette bis zum Stand 6 erzeugt hat und ein Angreifer aufbauend auf Block 1 des Netzwerkes selbst an einer alternativen Kette arbeitet. Diese soll die Komplexität der bisher gültigen Blockchain übertreffen und somit nach den Regeln des Blockchain-Netzwerkes übernommen werden, wobei im Falle der Abbildung die Blöcke 2 bis 6 des Mining-Netzwerkes durch die alternativen Blöcke des Angreifers ersetzt werden.

4.3.3 Abwehrmechanismen

Um Angriffe abzuwehren gibt es verschiedene Herangehensweisen. Grundsätzliche Konventionen, an die sich alle Teilnehmer halten sollten, implementierte Sicherheitsmechanismen im Code der Miner und Architektur Entscheidungen in Form des Proof-of-Konzepts. Im Folgendem sollen Absicherungen gegen *die Doppelausgabe* und *die nachträgliche Veränderung* beschrieben werden.

Die Doppelausgabe

Die Akzeptanz einer als doppelt ausgegebenen Transaktion als Zahlungsempfänger lässt sich ohne technische Lösung vermeiden, indem auf weitere Blöcke der Miner gewartet wird, die nach dem relevanten Block entstehen. Sollte so wie in Abbildung 18 eine Blockchain zustande gekommen sein, in welcher drei Blöcke nach der Zahlungstransaktion enthalten sind und die Transaktion in für den Empfänger korrekter Form enthalten ist, kann der Empfänger eine Gegenleistung erbringen. Andernfalls sind die Ungültigkeit der zunächst empfangenen Transaktion und damit auch der Betrugsversuch bewiesen.

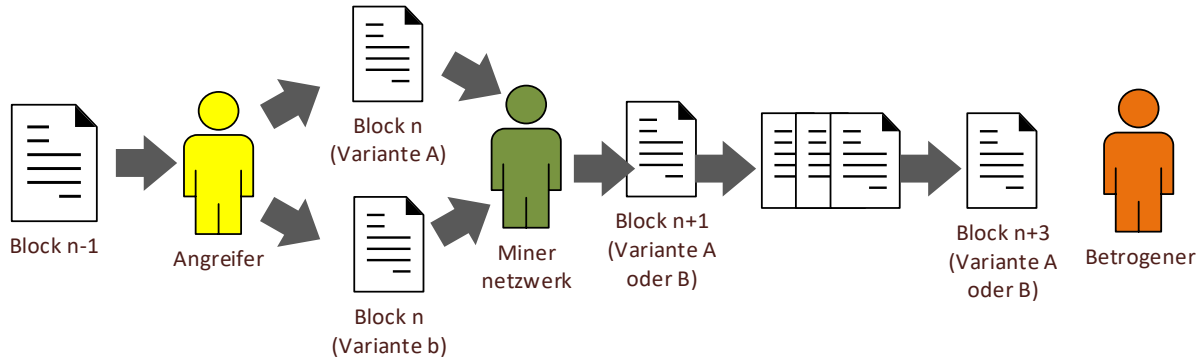

Abbildung 18: Gescheiterte Doppelausgabe

Eine technische Abwehrmöglichkeit stellt der Einsatz des Proof-of-Work da. Aufgrund der dafür benötigten Rechenleistung ist es für einen Miner bereits schwer, einen Block mit einer korrekten Transaktion zu erstellen, welcher als erster die generelle Blockkette weiterführt und somit in ihr aufgenommen wird. Die gleichzeitige Erstellung mehrerer Blöcke kann somit nahezu ausgeschlossen werden. Der Einsatz von Proof-of-Stake beispielsweise hat diesen Schutz nicht, da hier keine Rechenleistung erforderlich ist. Die konsequente Einhaltung nicht technischer Konventionen ist somit zwingend notwendig.

Die nachträgliche Veränderung

Die Möglichkeit einer Änderung der Blockchain ist maßgeblich von dem verwendeten Proof-of-Konzept abhängig, welches die dafür notwendigen Voraussetzungen bestimmt. Zudem besteht die Möglichkeit Blöcke als Meilensteine direkt im Source Code zu hinterlegen und damit zu finalisieren. Eine Änderung vor einem solchen Meilenstein ist somit ausgeschlossen, wodurch die Tragweite einer möglichen Änderung eingedämmt wird.

Die grundsätzliche Abwehrmethode unter Einsatz des PoW ist die Einbeziehung der Komplexität, also der kumulierten Schwierigkeit des PoW der aktiven und der alternativen Blockchain. Somit wird verhindert, dass eine längere Chain übernommen wird, welche in wenigen Minuten mit geringstmöglicher PoW-Schwierigkeit erstellt wurde. Da die Anerkennung einer alternativen Blockchain also die gleiche Blockchain-Komplexität erfordert, muss ein Angreifer auch über die Rechenkapazität verfügen, die die Erstellung einer Blockchain von gleicher Schwierigkeit erfordert. Er muss somit über eine dem Mining Netzwerk gleichende Rechenkapazität verfügen, um eine 50-prozentige Chance zu haben, nur den nächsten Block zu erstellen. Eine rückwirkende Veränderung, welche das Überhohlen der Chain erfordert, bedingt also einen Großteil der Gesamtrechenleistung des Mining-Netzwerkes. Je nach Netzwerkgröße kann dies eine immense oder aber geringe Hürde sein.

Die Abwehrmethode unter Einsatz des PoS unterscheidet sich dahingehend, dass die Rechenkapazität eines Angreifers irrelevant ist. Nur die ihm zur Verfügung stehenden Coins sind von Bedeutung. Eine kontinuierliche Überstimmung aller Miner, die im Rahmen einer

rückwirkenden Änderung notwendig ist, erfordert deshalb mehr als die Hälfte der sich im Umlauf befindenden Coins. Diese Hürde kann bereits in einem kleinen Netzwerk nahezu unüberwindbar sein, mit wachsendem Handelsvolumen steigt die Hürde noch weiter an.

4.3.4 Versuch: Angriff auf den Prototyp (Proof-of-Work)

Nach theoretischer Betrachtung möglicher Angriffsszenarien soll ein Angriff auf den Prototyp Aufschluss über die Erfolgsmöglichkeit eines wesentlichen Änderungsversuchs einer noch jungen Blockchain[6] geben. Dazu soll eine Testumgebung aufgebaut werden, in welcher sich drei Miner befinden, wobei einer die Rolle des Angreifers einnimmt. Gleichsam wurde eine Visualisierung implementiert, die den aktuellen Stand der drei Konkurrierenden Miner anzeigt und schließlich auch den Erfolg oder Misserfolg eines Angriffs darstellt. Damit eine Visualisierung gut beobachtbar und nachvollziehbar ist, ist eine gewisse Dauer der Blockgenerierung der einzelnen Miner von Nöten. Diese wird bedingt durch die eingesetzten Hash Verfahren nicht erreicht, sodass eine zufällige Verzögerung zwischen 0 und 120 Sekunden[7] innerhalb der Block Generierung eingebaut wurde. Die Rechenleistung der Teilnehmer spielt somit innerhalb der Versuchsumgebung eine untergeordnete Rolle.

Die Visualisierung

Als Visualisierungseinheit dient eine Java-Applikation, welche eine Nachrichtenempfänger-einheit in Form eines stets aktiven *ServerSockets* und eine grafische Oberfläche in Form einer *JavaFX Stage* enthält. Dazwischen befindet sich eine Interpretationseinheit, welche die von den Minern eingehenden Log-Einträge anhand ihres vordefinierten Codes erkennt. Die somit identifizierbaren mitgegebenen Informationen werden ausgelesen und schließlich der Anzeigeeinheit übergeben. Der konkrete Aufbau ist als Klassendiagramm in Anhang G Abbildung 37 zu finden.

Zudem wurden die Miner um eine Schnittstelle in Form eines *AppenderSkeleton* erweitert, sodass jeder Log-Eintrag mittels *DataOutputStream* an die zentrale Visualisierungseinheit versendet werden kann. (Anhang J RemoteAppender.java) Beim *AppenderSkeleton* handelt es sich um eine Komponente der Log4J Bibliothek, in welcher jeder Log-Aufruf innerhalb einer Anwendung als Ereignis auftritt und einen Methodenaufruf zufolge hat. Dieser wurde mit eigener Logik implementiert, die zum einen aus dem Auslesen des konkreten Log-Eintrags in einen String und zum anderen in der Aufnahme und dem Schließen einer Socket-Verbindung mit der Visualisierungseinheit zwecks Übertragung des Strings besteht.

---

[6] Eine Chain an deren Miningprozess erst wenige Nodes beteiligt sind (< 100 Miner).

[7] Vgl. Anhang J BlockGenerator.java – Zeile 53 & 30.

Der Angreifer

Als Angreifer dient eine leicht erweiterte Form des prototypischen Miners. So wird die Konfigurationsdatei um die Einträge *attacker* (boolean) und *speed* (int) erweitert und innerhalb der Programminitialisierung eingelesen. Die Annahme von Blöcken anderer Miner wird abhängig vom Wert *attacker* gemacht, wobei ein Angreifer (*attacker = true*) eingehende Blöcke ignoriert und somit stets sein Ziel verfolgt die eigene Blockchain durchzusetzen. Der Eintrag *speed* wird als Divisor der zufälligen Verzögerungszeit genutzt, wodurch eine Beschleunigung des Angreifers gegenüber den regulären Minern ermöglicht wird.

Durchführung

Die beiden regulären Miner sollen eine neue Blockchain erstellen. Sie erhalten bei einer Geschwindigkeit von 1 (speed=1) einen Vorsprung von 10 Minuten. Danach wird auch der Angreifer gestartet, wobei dieser eine Geschwindigkeit von 3 annimmt. Da er somit schneller ist als das Gesamtnetzwerk, welches über eine kumulierte Geschwindigkeit von 2 verfügt, sollte er im Stande sein innerhalb von 20 Minuten über eine ebenso lange Blockchain zu verfügen.[8] Nachdem der Angreifer über die längste Chain innerhalb des Netzwerk verfügt, wird die Übernahme dieser Kette seitens der Miner erwartet.

Ergebnis

Nach 10 Minuten verfügte die verteilte Blockchain über 17 Blöcke. Anhang I Abbildung 40 zeigt den vom Viewer angezeigten Stand nach 16 Minuten. Die ersten beiden Zeilen zeigen dabei den Status der regulär arbeitenden Miner 1 (Gelb) und 2 (Blau), die dritte Zeile wiederum die Chain des Angreifers (Pink), welcher zu diesem Zeitpunkt noch 7 Blöcke zurück lag. In Abbildung 41 liegen alle Miner gleichauf bei einer Kettenlänge von 31 Blöcken nach 20 Minuten. Eine Minute später konnte der Angreifer schließlich das Netzwerk mit seinen Block 33 überholen, um Miner 1 und 2 nacheinander den eigenen Stand erfolgreich anbieten zu können, wodurch alle bisherigen Blöcke seitens der ehrlichen Miner verworfen werden mussten. (Abbildung 42, Abbildung 43)

Dieses Angriffsszenario verdeutlicht neben der korrekten Funktion des Prototyps vor allem die Verwundbarkeit einer auf Proof-of-Work basierenden Blockchain, welche über nur geringe Rechenkapazitäten verfügt. Dabei kann der Test durchaus in die Realität übertragen werden,

[8] Angreifer hat eine Geschwindigkeit von 3, wodurch er zwischen 0 und 40 Sekunden/Blockerstellung benötigt. Bei durchschnittlich 20 Sekunden/Blockerstellung hätte er somit in 20 Minuten 60 Blöcke generiert.

Miner haben eine kumulierte Geschwindigkeit von 2, wodurch sie zwischen 0 und 60 Sekunden/Blockerstellung benötigen. Bei durchschnittlich 30 Sekunden/Block hätten sie somit in 30 Minuten 60 Blöcke generiert.

wo der Verbund GHash bereits am 12. Juni 2012 „über zwölf Stunden mehr als 51 Prozent der Rechenleistung für Bitcoins gestellt [hat]. (Ernst, 2014)“ Somit bestand bereits auch im Großen die Möglichkeit einer solchen Manipulation.

4.4 Kritische Reflexion

Der im Rahmen dieser Arbeit erstellte Prototyp kann als Basis einer Blockchain genutzt werden. Für die Entwicklung zur vollwertigen Kryptowährung bedürfte es dabei noch Erweiterungen, in Form von komplexen Hash-Verfahren, Authentizitätsprüfungen, alternative Herangehensweisen zur Findung eines PoW und Optimierungen in der Netzwerk Kommunikation. Dennoch wird er seinen funktionalen Anforderungen gänzlich gerecht, indem er die Grundmechanismen einer für den realen Einsatz tauglichen Blockchain beherrscht. Gemäß den Anforderungen können Chains erstellt werden, wobei eingehende Transaktionen, zwischengespeichert in einer Queue, sicher in Blöcken erfasst werden. Blöcke (und bei Bedarf Blockreihen) können an andere Miner versendet und eingehende Blöcke anderer Teilnehmer interpretiert und ggf. aufgenommen werden. Manipulationen der Chain werden dabei durch Integritätsprüfungen erkannt. Auch die Bekanntmachung der Nodes innerhalb eines geschlossenen Netzwerkes mittels Broadcasts ist ohne einen weiteren Dienst möglich.

Die nicht funktionalen Anforderungen wurden mittels einer Wallet-Anwendung realisiert. Mit ihr können Transaktionen versendet und der aktuelle Kontostand sowie auf die eigene Adresse gerichtete Transaktionen angezeigt werden. Zudem kann der Status mehrerer Miner mittels Visualisierungsanwendung angezeigt und untersucht werden.

Neben den analysierten Angriffsmöglichkeiten, welche je nach Ausgangsbedingungen keine oder schwerwiegende Schäden verursachen können, verdeutlicht die Entwicklung des Prototyps weitere Problematiken der Technologie. Die umfangreiche Netzwerk Kommunikation lässt das Ausmaß des Kommunikationsaufwandes einer Blockchain erkennen, bei der der ständige Abgleich der je Miner geführten Blockchain zur Notwendigkeit mehrerer gleichsam offen gehaltenen Socket-Verbindungen führt. Eine Limitierungen des Bitcoin-Netzwerkes, durch die maximal 7 Transaktionen pro Sekunde aufgenommen werden können, ist die Folge (Swan, 2015, S. 83).

Unter Berücksichtigung der Schwächen, welche besonders wie im Beispielangriff zu sehen noch junge Kryptowährungen in kleinen Netzwerken betreffen, kann dennoch eine sichere und gleichsam ressourcenschonende Implementation entstehen. Hier kann der Einsatz hybrider Proof-of-Konzepte zu einer zuverlässigen und Ressourcen schonenden Blockchain führen.

Eine solche zuverlässige Technologie kann dann um weitere Informationen erweitert werden, sodass nicht nur eine Funktion als Kryptowährung, sondern auch als Datenspeicher möglich

wird. Innerhalb dieser transaktionsgebundenen Speichereinheiten können nicht nur einfache Dokumente, sondern auch ausführbare Programme gespeichert und innerhalb einer in den Minern laufenden virtuellen Umgebung ausgeführt werden. Diese Programme dienen in heutigen Anwendungsfällen als Smart Contracts.

5 Fachliche Konzeption eines Smart Contracts

In diesem Unterkapitel wird die Geschäftsidee vorgestellt, um den in Kapitel 2.2 vorgestellten Smart Contracts einen praktischen Nutzen zu geben. Die Geschäftsidee ist in der Finanzbranche beheimatet, genauer gesagt im Derivathandel. Aus diesem Grund wird zunächst eine Einführung in den Derivathandel gegeben. Daraufhin folgt die Konkretisierung des Geschäftsvorfalls, die Abbildung des Derivathandles in einer dezentralen Anwendung. Zum Abschluss des Unterkapitels werden praktische und rechtliche Einschränkungen der Idee betrachtet.

5.1 Einführung in den Derivathandel

Wenn Unternehmen Geschäfte abschließen, deren Erfüllungspflicht in der Zukunft liegt, entsteht häufig ein finanzielles Risiko für das Unternehmen. Dieses Risiko ist durch die schwankenden Marktpreise von Rohstoffen, Zinsen, Währungen, Aktien und weiteren materiellen und nicht materiellen Gütern begründet (HeldtB).

Ein kurzes Beispiel soll diese Risiken verdeutlichen: So verpflichtet sich Beispielsweise ein deutsches Unternehmen vertraglich dazu, in einem Jahr Waren an ein US-amerikanisches Unternehmen zu liefern. Das amerikanische Unternehmen verpflichtet sich im Gegenzug dazu, den heute vereinbarten Rechnungsbetrag in einem Jahr in US-Dollar zu bezahlen. Mit der Unterzeichnung des Vertrages geht das deutsche Unternehmen ein Währungsrisiko ein, da der Wechselkurs von Euro und Dollar Schwankungen unterliegt (Schöning).

Zur Kompensation dieser finanziellen Risiken können Derivatverträge abgeschlossen werden. Genauer gesagt einen Forward-Kontrakt mit Cash-Settlement. Ein Forward-Kontrakt ist eine Vereinbarung ein bestimmtes Gut zu einem bestimmten Zeitpunkt, zu einem bestimmten Preis abzunehmen (Hull, 2015, S. 29). Cash Settlement bedeutet in diesem Fall, dass nicht das Gut selber abgenommen bzw. geliefert wird, sondern nur eine Ausgleichszahlung zum veränderten Marktpreis stattfindet (HeldtA).

Die Unterzeichnung eines Derivatvertrages bedeutet ein weiteres finanzielles Risiko für die beteiligten Unternehmen. Das Solvenzrisiko, dass einer der beteiligten Parteien seiner Zahlungsverpflichtung nicht nachkommen kann. Um dieses Risiko zu mindern, wird häufig eine unabhängige dritte Partei eingeschaltet – das sogenannte Clearinghouse. Das Clearinghouse ist ein Treuhänder, welcher die Margin Konten der Vertragsparteien verwaltet (Hull, 2015, S. 57 ff).

Die nachfolgende Tabelle soll die Arbeitsweise des Clearinghouses verdeutlichen. Es ist ein in Anlehnung an (Hull, 2015, S. 57 ff), angepasstes Beispiel. Die beiden Unternehmen A und B haben gegensätzliche Positionen eingenommen. Unternehmen A ist aus deutscher Sicht ein

exportierendes Unternehmen und erhält in einem Jahr eine Zahlung über 110.000 $, was zum jetzigen Zeitpunkt 100.000 € entspricht. Unternehmen B ist aus deutscher Sicht ein importierendes Unternehmen, welches in einem Jahr eine Rechnung über 110.000 $ begleichen muss. Da beide Unternehmen in der internen Kostenrechnung mit dem aktuellen Wechselkurs (1 € = 1,10 $) gerechnet haben, wollen sie diesen Wechselkurs über einen Derivatvertrag absichern. Um kein Solvenzrisiko einzugehen, entschließen sich beide Unternehmen den Derivatvertrag über ein Clearinghouse abwickeln zu lassen.

Zum 1. Januar müssen die Unternehmen jeweils eine Sicherheitsleistung (Margin) in Höhe von 10.000 € an das Clearinghouse überweisen. Das Clearinghouse führt für die beiden Unternehmen jeweils ein internes Konto, auf welchem sich zum 1. Januar jeweils 10.000 € befinden. Die Aufgabe des Clearinghouse ist es, ein tägliches Settlement der beiden Konten anhand des tagesaktuellen Wechselkurs durchzuführen.

Zum 2. Januar beträgt der Wechselkurs nicht mehr 1 € = 1,10 $, sondern 1 € = 1,12 $. Diese Kursveränderung wäre ohne einen Derivatvertrag für das importierende Unternehmen B positiv, da es weniger Euro aufwenden muss, um seine Rechnung in Dollar zu bezahlen. Im Umkehrschluss wäre diese Kursveränderung für das exportierende Unternehmen B negativ, da es für die erhaltenen Dollar weniger Euro erhält. Durch den Derivatvertrag verzichten beide Unternehmen jedoch auf Gewinne aus für sie positiven Kursveränderungen, sowie auf Verluste aus für sie negativen Kursveränderungen. Aus diesem Grund gleicht das Clearinghouse die Konten der beiden Vertragsparteien genau um die Höhe der entstandenen Kursgewinne bzw. Kursverluste aus. In diesem Fall wird das Konto von Unternehmen B mit 1785,71 € belastet und dem Konto von Unternehmen A gutgeschrieben.

Am 3. Januar entwickelt sich der Kurs in die entgegengesetzte Richtung, sodass das Konto von Unternehme A mit 3.637,57 € belastet wird und dem Unternehmen B gutgeschrieben wird.

Diesen Ausgleich führt das Clearinghouse nun bis zum Ende der Vertragslaufzeit weiter durch. Am 12. März entsteht jedoch die Besonderheit, dass das Margin Konto des Unternehmens B unterhalb der Maintenance Margin von 5000 € sinken würde. Das Unternehmen B überweist daraufhin weitere 5000 € auf sein Konto beim Clearinghouse. Würde das Unternehmen B der Zahlungsaufforderung nicht nachkommen, so würde der Derivatvertrag aufgelöst und beide Parteien in der Höhe ihrer Kontostände ausbezahlt. Das Unternehmen A müsste sich in diesem Fall einen anderen Vertragspartner suchen, um dem Wechselkursrisiko weiterhin entgegen zu wirken.

Dieser Prozess aus täglichem Ausgleich und eventuellen Nachzahlungen in Form von Margin Calls setzt sich bis zum 31. Dezember fort, sofern die beiden Unternehmen den möglichen Zahlungsaufforderungen nachkommen. An diesem Tag werden den beiden Unternehmen die verbleibenden Beträge der Konten ausbezahlt.

Datum	Unt. A	Unt. B	Kurs	Summe Euro	Änderung	Margin Call
01. Jan	10.000,00 €	10.000,00 €	0,9091 €	100.000,00 €		
02. Jan	11.785,71 €	8.214,29 €	0,8929 €	98.214,29 €	1.785,71 €	
03. Jan	8.148,15 €	11.851,85 €	0,9259 €	101.851,85 €	- 3.637,57 €	
			.			
			.			
			.			
12. Mrz	16.779,66 €	8.220,34 €	0,8475 €	93.220,34 €	8.631,51 €	5.000 Unt. B
13. Mrz	17.563,03 €	7.436,97 €	0,8403 €	92.436,97 €	783,36 €	
			.			
			.			
			.			
29. Dez	18.333,33 €	6.666,67 €	0,8333 €	91.666,67 €	770,31 €	
30. Dez	21.290,32 €	8.709,68 €	0,8065 €	88.709,68 €	2.956,99 €	5.000 Unt. B
31. Dez	22.000,00 €	8.000,00 €	0,8000 €	88.000,00 €	709,68 €	

Tabelle 2: Beispiel Clearinghouse

Betrachtet man die Gesamtrechnung, so ist festzustellen, dass Unternehmen A zu Beginn des Clearingvertrags 10.000 € an das Clearinghouse bezahlt hat und zum Ende der Vertragslaufzeit 22.000 € für den veränderten Wechselkurs vom Clearinghouse erhält. Daneben erhält Unternehmen A die 110.000 $ aus seiner Geschäftstätigkeit. Dies entspricht mit dem tagesaktuellen Wechselkurs verrechnet 88.000 €. Somit erhält Unternehmen A genau die 100.000 €, mit welchen es zum Zeitpunkt des Vertragsabschlusses gerechnet hat.

An Clearinghouse bezahlt:	10.000 €
Erhält von Clearinghouse:	22.000 €
Erhält Dollar 110.000 $ * (1/1,25):	88.000 €
Effektiv erhalten:	100.000 €

Tabelle 3: Endrechnung Unternehmen A

Unternehmen B hingegen hat über die Initial Margin von 10.000 € und zwei Margin Calls a 5.000 € insgesamt 20.000 € an das Clearinghouse überwiesen. Für den veränderten Wechselkurs erhält es am Ende der Laufzeit des Derivatvertrags 8.000 € vom Clearinghouse zurück. Für die Begleichung seiner Rechnung über 110.000 $ muss Unternehmen B 88.000 € aufwenden. Somit hat Unternehmen B insgesamt die kalkulierten 100.000 € bezahlt.

An Clearinghouse bezahlt:(Margin)	10.000 €
An Clearinghouse bezahlt (Margin Call):	5.000 €
An Clearinghouse bezahlt: (Margin Call):	5.000 €
Erhält von Clearinghouse:	8.000 €
Bezahlt Dollar: 110.000 $ * (1/1,25)	88.000 €
Effektiv bezahlt:	100.000 €

Tabelle 4: Endrechnung Unternehmen B

Bei der Berechnung sind Gebühren für das Clearinghouse sowie Zinsbelastungen für das gebundene Kapital durch die Margins bewusst nicht einberechnet worden, da sie für die Absicherung des Marktpreises nur eine untergeordnete Rolle spielen. Weiterhin wird nicht betrachtet, dass die aktuellen Spot Kurse und die für Derivatverträge verwendeten Forward Kurse in der Regel geringe Unterschiede aufweisen (Hull, 2015, S. 28).

5.2 Idee

Die Idee für den Geschäftsvorfall ist es, den in Kapitel 2.2 beschriebenen Nutzen und die Möglichkeiten einer dezentralen Anwendung für das Clearing eines Derivatvertrags zu nutzen. Genauer gesagt soll das Clearinghouse durch eine dezentrale Anwendung auf Basis der Blockchain-Technologie ersetzt werden.

Der wesentliche Grund, weshalb die beiden Vertragsparteien eines Derivatvertrags auf ein Clearinghouse zurückgreifen, ist mangelndes Vertrauen der Vertragsparteien gegenüber der jeweils anderen Partei. Dieses unzureichende Vertrauen ist primär mit der unbekannten oder nur schwer einzuschätzenden Zahlungsfähigkeit des anderen Vertragspartners begründet. Daher wird in der Regel eine unabhängige dritte Partei eingeschaltet, welcher beide Parteien ein höheres Vertrauen entgegenbringen – das Clearinghouse. (Hull, 2015, S. 58)

Die Hinzunahme eines Clearinghouses bringt für die beiden Vertragsparteien den Vorteil, dass sie nicht in die Solvenz des anderen Vertragspartners vertrauen müssen. Daneben müssen die Vertragsparteien keine Ressourcen für das Clearing im eigenen Haus vorhalten.

Durch die Hinzunahme eines unabhängigen Dritten entstehen jedoch auch Nachteile, so müssen Gebühren für das Clearing bezahlt werden. Außerdem muss nicht mehr in die Solvenz des Vertragspartners vertraut werden, jedoch in die Solvenz des Clearinghouses. (Hull, 2015, S. 59)

Zur besseren Verdeutlichung der Geschäftsidee wurde ein BPMN-Diagramm erstellt. Nachfolgend werden Ausschnitte dieses Diagramms erläutert. Das gesamte Diagramm ist im

Anhang N zu finden. Beim BPMN-Diagramm, ebenso wie beim späteren Prototyp ist der Fokus auf eine protypische Implementierung gelegt. Daher werden etwaige Sonderfälle wie die Ablehnung des Vertragsangebots von einer Partei oder das Ausbleiben der Initial Margin nicht berücksichtigt.

Im ersten Abschnitt wird das Zustandekommen des Clearingvertrages beschrieben. Da bei einem Smart Contract keine zentrale Instanz vorhanden ist, muss das Erstellen des Clearingvertrags von einer der beiden Parteien übernommen werden. Zur Vereinfachung des Ablaufs soll der Vertrag immer vom Käufer eines Wirtschaftsgutes erstellt werden.

Nach der Erstellung des Contracts werden beide Parteien zum Einzahlen der Initial Margin aufgefordert. Sobald die Initial Margin von beiden Parteien eingezahlt wurde beginnt der eigentliche Prozess des Clearings, welcher im nächsten Abschnitt dargestellt wird.

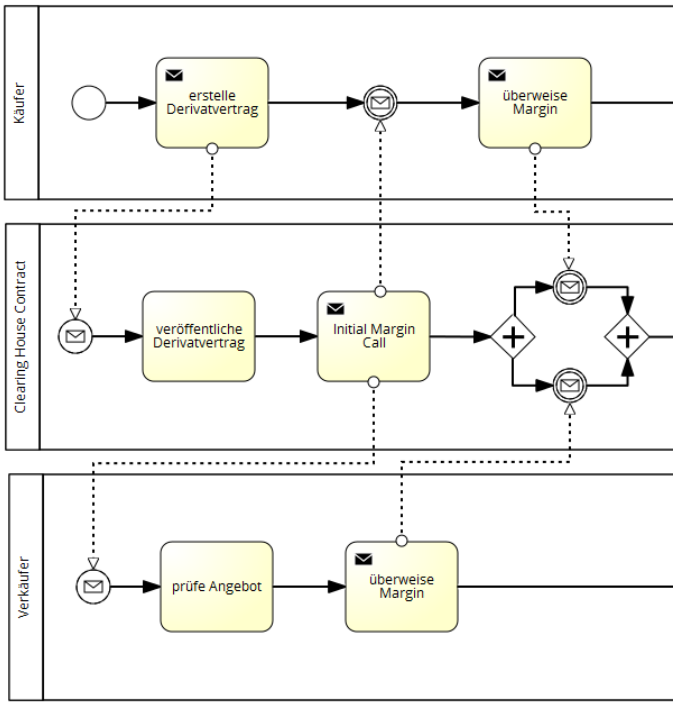

Abbildung 19: BPMN-Diagramm des Geschäftsvorfalls Abschnitt 1

Der zweite Abschnitt beginnt damit, dass das Clearing täglich angestoßen wird. Der zugeklappte Unterprozess „Ausgleich der Konten (Clearing)“ wird im nachfolgenden Abschnitt detailliert erläutert. Nach dem Ausgleich der Konten wird geprüft, ob der Kontostand von einer der beiden Vertragsparteien unterhalb der Maintenance Margin liegt. Ist dies der Fall, so wird die Partei zu einem Margin Call aufgefordert. Die aufgeforderte Partei muss entscheiden, ob sie der Aufforderung nachkommt oder die Frist verstreichen lässt. Sollte es zu keiner Aufforderung zum Zahlen einer Margin kommen oder die Margin wird fristgerecht bezahlt, so wird gegen Ende dieses Ausschnitts geprüft, ob das Enddatum des Kontraktes erreicht wurde. Ist das Enddatum noch nicht erreicht wird der Teilausschnitt am nächsten Tag

wiederholt. Ist das Enddatum erreicht oder einem Margin Call wird nicht nachgekommen, so wird das Clearing beendet und der Käufer und der Verkäufer ausbezahlt. Dies wird im 3. Abschnitt besprochen.

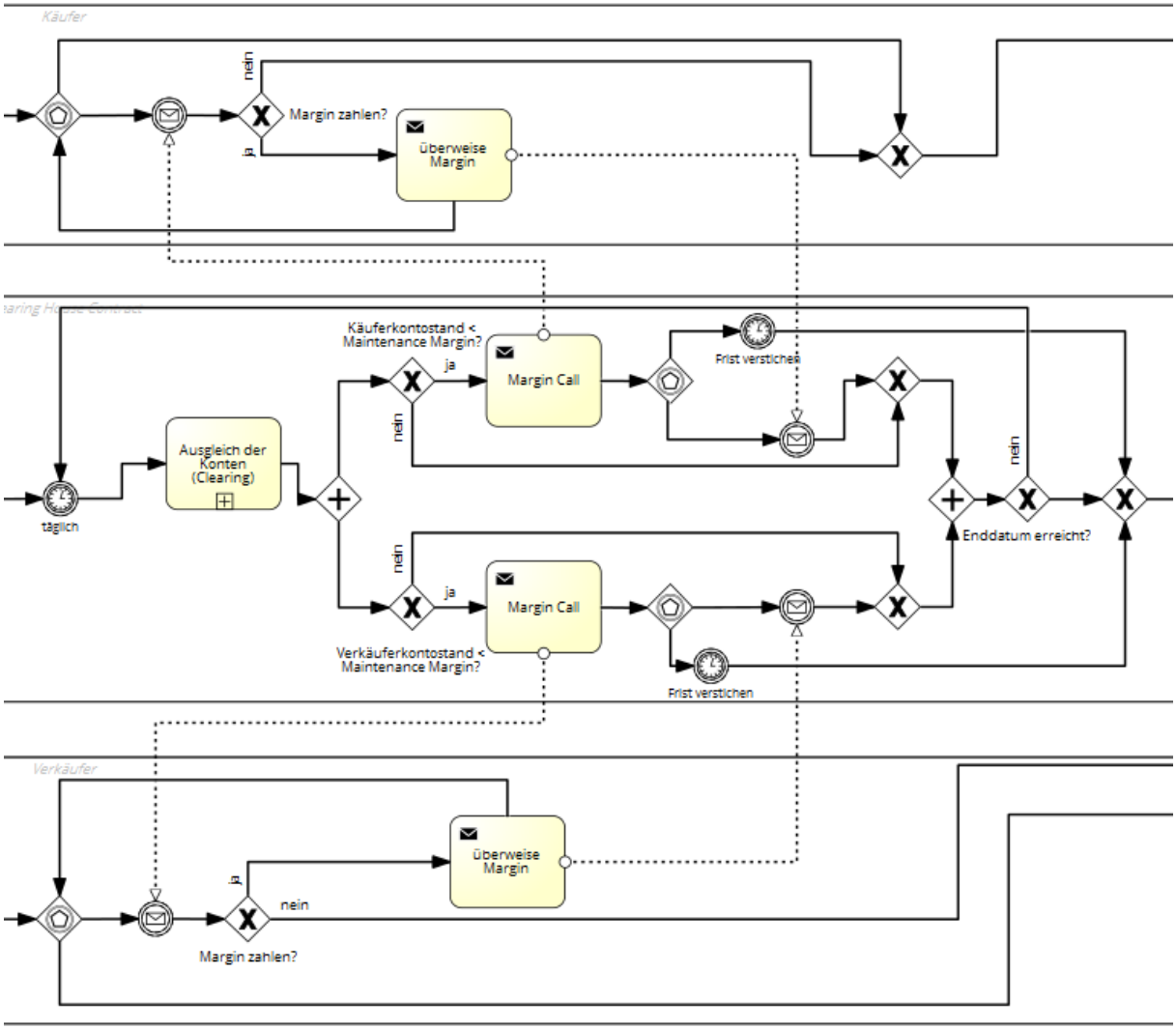

Abbildung 20: BPMN-Diagramm des Geschäftsvorfalls Abschnitt 2

In diesem Abschnitt wird der zugeklappte Unterprozess „Clearing“ beschrieben. Zunächst muss der Smart Contract die Marktdaten des entsprechenden Wirtschaftsgutes von einem externen Markdaten Provider beziehen. Daraufhin wird die Änderung der Marktdaten zum Vortag berechnet. Für die Berechnung wird folgende Formel verwendet:

$$\Delta=(KG-KP_{t-1})-(KG-KP_t)$$

Δ = Änderung

KG= Kontraktgröße

KP_t= Kontraktpreis am Tag t

Ist die Änderung positiv, so wird das Margin Konto des Käufers um den Änderungsbetrag reduziert und das Konto des Verkäufers um denselben Betrag erhöht. Bei einer negativen Änderung wird das Konto des Verkäufers um den Änderungsbetrag reduziert und das des Käufers um den Betrag erhöht. Nach diesem Schritt ist der Unterprozess des Clearings beendet.

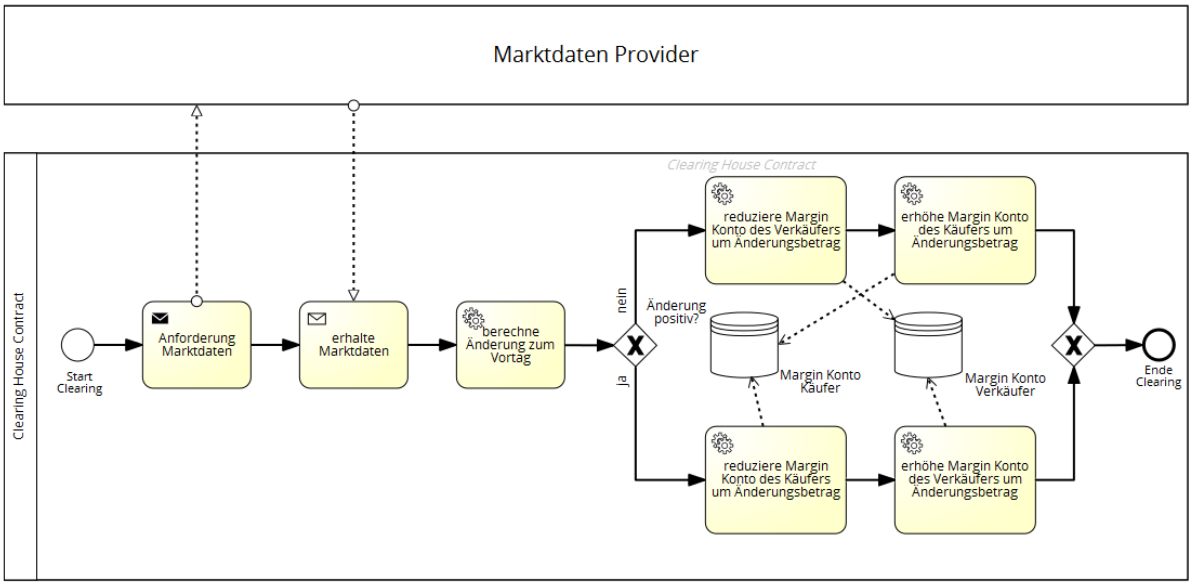

Abbildung 21: Clearing im Smart Contract

Im letzten Abschnitt wird, bevor das Settlement erfolgt, noch geprüft, ob einer der beiden Kontostände im negative Bereich ist. Ist das der Fall, so muss der Kontostand der anderen Partei um diesen Betrag reduziert werden. Siehe zu diesem Sonderfall auch den Abschnitt „Absicherung bei starken Kursveränderungen“ im Kapitel 5.3. Der Aufgeklappte Unterprozess „prüfe Kontostände“ befindet sich im Anhang O.

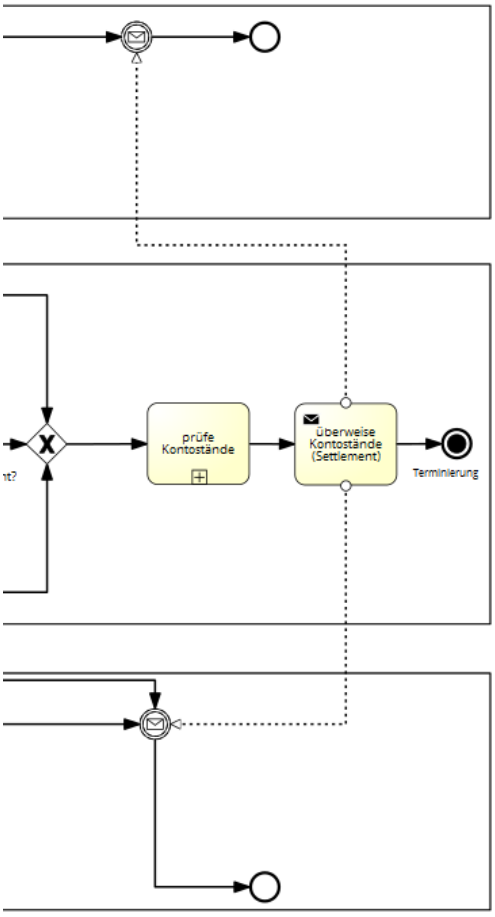

Abbildung 22: BPMN-Diagramm des Geschäftsvorfalls Abschnitt 3

5.3 Rechtliche und praktische Einschränkung

Die Volatilität der zum Clearing eingesetzten Kryptowährung

Beim Clearing mit einem Smart Contract müssen Gegenwerte im Smart Contract hinterlegt werden. Diese Gegenwerte sind bei Smart Contracts typischerweise Kryptowährungen, wie die in Kapitel 2.1 genannten Bitcoin oder die für Smart Contract Plattform Ethereum geschaffenen Ether. Diese Kryptowährungen unterliegen zum jetzigen Zeitpunkt großen Kurschwankungen. Betrachtet man den Wechselkurs zwischen Bitcoin und Euro der letzten 90 Tage (bezogen auf den 26.01.2016), so lag der Minimalwert der Zeitspanne am 28. Oktober 2015 bei 277,90 € für einen Bitcoin. Der Maximalwert wurde am 18. Dezember 2015 mit 426,78 € für einen Bitcoin erreicht. Dies entspricht einer maximalen Kursänderung von 53,57 %. (CoingeckoA)

Ähnliche Beobachtungen können bei der Betrachtung des Wechselkurses zwischen Euro und Ether gemacht werden. Hier wurde der Maximalwert der letzten 90 Tage (bezogen auf den 26.01.2016), am 26. Januar 2016 mit 2,54 € für einen Ether erzielt. Der Minimalwert der Zeitspanne betrug am 09. Dezember 2015 0,72 € für einen Ether. Dies entspricht einer maximalen Kursänderung von 352,78 %. (CoingeckoB)

Diese stark volatilen Kryptowährungen sind ungeeignet, um als Gegenwert beim Clearing eingesetzt zu werden, da genau durch das Clearing von Derivatverträgen Schwankungen in Marktpreisen ausgeglichen werden sollen.

Da es sich bei der Entwicklung des Prototyps um eine Machbarkeitsstudie handelt, kann dieser Aspekt vernachlässigt werden. Sollte es zu einer praktischen Umsetzung kommen, so sollte geprüft werden, ob die Möglichkeit besteht, weniger volatile Gegenwerte im Smart Contract zu hinterlegen.

Absicherung bei starken Kursveränderungen

Der auf einen Smart Contract übertragene Geschäftsvorfall entspricht einem außerbörslichem (Over the Counter) gehandeltem Forward, bei welchem das Kreditrisiko auf eine zentrale Clearingstelle übertragen wird (Hull, 2015, S. 61). Im Falle einer starken Kursveränderung kann es dazu kommen, dass das Margin Konto einer Partei in den negativen Wertebereich gerät. Sollte die betroffene Partei daraufhin ihrer Zahlungsverpflichtung in Form eines Margin Calls nicht nachkommen, so besteht für die Clearingstelle die Pflicht die andere Partei aus eigenen Mitteln zu bezahlen (Hull, 2015, S. 59). In diesem Fall ist die zentrale Clearingstelle ein Smart Contract und keine eigenständige juristische Person, welche über ein eigenes Vermögen verfügt (Berwanger). Daher bleibt bei der Übertragung des Geschäftsvorfalls in einen Smart Contract ein gewisses Kreditrisiko bestehen.

Meldepflicht für Derivatverträge

Nach Artikel 9 Absatz 1 der europäischen Verordnung Nr. 648/2012 müssen Derivatkontrakte bei Abschluss, Änderung oder Beendigung an ein registriertes und anerkanntes Transaktionsregister gemeldet werden. Auf diese Verpflichtung wird bei der prototypischen Entwicklung ebenfalls nicht eingegangen, da über den Prototyp nie reale Derivatverträge abgewickelt werden sollen.

6 Smart Oracles

Als Vorarbeit zur Umsetzung des Geschäftsvorfalles mit einem Smart Contract, muss die Besonderheit betrachtet werden, dass Smart Contracts ein in sich geschlossenes System sind und keine Verbindung zur Außenwelt aufbauen können.

In diesem Kapitel wird zunächst die Problemstellung erläutert und daraufhin verschiedene Lösungsansätze vorgestellt. Eine prototypische Implementierung eines Smart Oracles erfolgt in Kapitel 7.3 im Rahmen der Gesamtimplementierung des Geschäftsvorfalls.

6.1 Problemstellung

Smart Contracts können von sich aus keine Verbindung zur Außenwelt aufbauen und daher auch keine externen APIs von Marktdatenprovider wie der Europäischen Zentralbank, Onvista.com oder Yahoo Finance ansprechen. Die einzige Möglichkeit externe Daten in die geschlossenen Blockchain zubekommen, besteht darin, die Smart Contracts innerhalb der Blockchain über Transaktionen anzusprechen (Ethereum, 2015).

Die naheliegende Lösung für den in Kapitel 5 gewählten Geschäftsvorfall wäre es, eine der beiden Vertragsparteien mit der Versorgung der Marktdaten über Transaktionen zu betrauen. Dies würde jedoch bedeuten, dass diese Partei große Macht besitzt, den Kontrakt zu ihren Gunsten zu manipulieren. Daher scheidet dieser einfache Ansatz aus.

Die Erweiterung des ersten Ansatzes würde darin bestehen, dass beide Vertragsparteien den Kontrakt mit Marktdaten über Transaktionen versorgen. Dies würde die Manipulation durch eine der beiden Parteien verhindern. Jedoch würde dem Kontrakt die Entscheidungsgrundlage fehlen, wenn die Marktdaten der beiden Parteien voneinander abweichen. Daher scheidet diese Möglichkeit ebenfalls aus.

6.2 Lösungsansätze

In der noch jungen Welt, der auf der Blockchain-Technologie beruhenden Smart Contracts, hat sich der Begriff Smart Oracle etabliert wenn es darum geht, einen Zustand der realen Welt in einen Smart Contract einfließen zulassen (Ripple Labs, 2014).

Smart Oracles ermöglichen es der streng deterministischen Blockchain Technologie nicht deterministische Umweltzustände einfließen zu lassen. Da diese Umweltzustände, sobald sie einmal in der Blockchain integriert sind, nicht mehr abänderbar sind und streng deterministisch weiterverarbeitet werden, muss einem Smart Oracle großes Vertrauen entgegengebracht werden. (Ripple Labs, 2014)

Technisch gesehen ist ein Oracle ein einfacher Akteur, der wie die anderen Akteure, Transaktionen auf der Blockchain durchführen kann. In diesen Transaktionen codiert das

Oracle die Zustände der realen Welt. Die eigentlichen Smart Contracts haben Zugriff auf diese Daten sobald die Transaktion von der Blockchain verifiziert wurde, da alle Transaktionen auf der Blockchain öffentlich sind. Sieh dazu auch Kapitel 3.3.1.

Um das benötigte Vertrauen für ein Oracle aufzubauen gibt es verschiedene Ansätze. In den nachfolgenden Unterkapiteln werden drei Ansätze näher betrachtet und die Vor- und Nachteile der Lösungen aufgezeigt.

6.2.1 Distributed Oracles

Der erste Ansatz besteht, vereinfacht gesagt, darin nicht einem Smart Oracle zu vertrauen, sondern darin die Daten von mehreren Oracles zu beziehen und die Mehrheitsentscheidung der Oracles als Faktum anzuerkennen.

Das Startup-Unternehmen Orisi beschreibt in seinem auf Github veröffentlichen White Paper, die Funktionsweise von Distributed Oracles detaillierter (Orisi, 2014). Das Whitepaper beschreibt die Oracle Funktion für die Bitcoin Blockchain. Das Prinzip kann aber auch auf andere Blockchain basierte Smart Contracts angewendet werden.

Bei diesem Prinzip verständigen sich die Vertragsparteien eines Smart Contracts auf eine Anzahl von Smart Oracles, welchen sie ein gewisses Vertrauen entgegenbringen. Die gewählten Oracles versorgen den Smart Contract mit Daten aus der realen Welt. Der Smart Contract entscheidet dann anhand von vorher festgelegten Regeln, wie mit den Oracle Daten umgegangen wird. Diese Gestaltung dieser Regeln obliegt den Vertragspartnern des Smart Contracts. Bei booleschen Werten könnten sich die Vertragsparteien zum Beispiel darauf verständigen, dass eine Einfache- oder eine Zweidrittelmehrheit erreicht werden muss, damit der Wert als Faktum anerkannt wird. Bei Dezimalzahlen, wie sie etwa bei Kurswerten auftreten, empfiehlt es sich nicht auf die Einfache- oder Zweidrittelmehrheit zu vertrauen, da je nach Datenquelle der Oracles leichte Unterschiede, insbesondere in den Nachkommastellen, auftreten können[9]. Hier würde es sich empfehlen, den Mittelwert aus den gelieferten Daten zu bilden. Um hier die große Manipulation durch ein Oracle zu verhindern, könnten der Maximal- und Minimalwert aus der Berechnung ausgeschlossen werden.

Der Vorteil des Distributed Oracles besteht darin, dass ein Single Point of Failure ausgeschlossen wird. Es besteht nicht mehr die Gefahr, dass ein einzelnes Oracle gehackt wird, fehlerhaft arbeitet, abgeschaltet oder bewusst manipuliert wird (Orisi, 2014).

[9] Stichprobe zum Euro/Dollar Wechselkurs am 04.02.2016 um 13 Uhr: www.finanzen.net: 1,1174; www.onvista.de: 1,1179; http://www.ariva.de: 1,11718

Als Schwierigkeit, bei der Verwendung von Distributed Oracles, kann die Einigung auf die verwendeten Oracles zwischen den Vertragspartnern angesehen werden. Zum einen muss sich erstmal eine gewisse Anzahl an Oracles am Markt etablieren. Momentan befinden sich nur drei Oracles zu Testzwecken in der vom Startup Orisi betriebenen Liste (Orisi, 2016). Zum anderen ist Manipulationsgefahr weiterhin hoch, wenn es einem Vertragspartner gelingt Einfluss auf mehrere Oracles zu nehmen.

6.2.2 Reality Keys

Reality Keys ist ein Oracle Dienst der vom japanischen Unternehmen Social Minds Inc (KK) betrieben wird (Reality KeysA). Reality Keys ist, wie Orisi, auf die Bitcoin Blockchain ausgelegt. Es bietet aber auch eine Integration für andere Plattformen wie Ethereum und Eris an (Reality KeysB).

Die Besonderheit bei Reality Keys ist, dass sie eine Mischform aus einem automatischen Oracle und einem von Menschenhand kontrollierten Oracle betreiben. Im Normalfall arbeitet Reality Keys wie ein normales Oracle, indem es die benötigten Daten von einer öffentlichen API einfließen lässt und innerhalb der Blockchain zur Verfügung stellt. Für diesen Normalfall berechnet Reality Keys keine Gebühren. (Reality KeysA)

Sollten eine oder mehrere, der an dem Smart Contract beteiligten Parteien, Zweifel an der Richtigkeit der vom Oracle übertragenen Werte haben, so können sie daraufhin eine menschliche Nachprüfung beauftragen. Für die menschliche Nachprüfung berechnet Reality Keys eine Gebühr von 10 USD je zu prüfendem Wert. (Reality KeysC)

Das von Reality Keys angebotene Verfahren bietet den Vorteil, dass im Fall von falsch bereitgestellten Werten einer externe API, nicht erreichbaren APIs oder Fehlern beim Übertragungsweg eine menschliche Nachprüfung beauftragt werden kann und der Smart Contract nicht mit falschen Werten weiterarbeitet.

Nachteilig ist hingegen, dass der Smart Contract eine Art Fallback-Mechanismus implementieren muss, der eingreift wenn ein Wert von einer der Vertragsparteien als falsch deklariert wird und eine menschliche Prüfung beauftragt wird. Daneben sind die Vertragspartner nicht dagegen geschützt, dass Reality Keys seinen Dienst einstellt, dass ein Fehler bei der menschlichen Prüfung passiert oder es einem Vertragspartner gelingt den Dienst von Reality Keys auf technische oder menschliche Weise zu manipulieren.

6.2.3 Oraclize

Oraclize ist ein Oracle-Dienst, welcher sich selber als nachprüfbar ehrlich bezeichnet. Diese nachprüfbare Ehrlichkeit erreicht der Dienst durch die Verwendung von TLSNotary (Bertani,

2015). TLSNotary ist ein Dienst, welcher sich in den Verbindungsaufbau einer TLS-Verbindung schaltet, um zu bestätigen, dass ein bestimmter Server zu einem bestimmten Zeitpunkt bestimmte Daten versendet hat (TLSNotary, 2015). Oraclize bietet zum Zeitpunkt der Ausarbeitung eine Integration in die Bitcoin- und die Ethereum-Blockchain an.

Durch die Verwendung von TLSNotary erhöht das Oracle sein Vertrauensniveau. Eine vollständige Sicherheit bietet dieser Dienst jedoch auch nicht an, da ähnlich wie bei Reality Keys die Gefahr besteht, dass Oraclize seinen Dienst einstellt oder das System gehackt wird.

7 Prototypische Implementierung des Smart Contracts

In den nachfolgenden Unterkapiteln wird die prototypische Implementierung des in Kapitel 5.2 entwickelten Geschäftsvorfall beschrieben. Dazu wird zuerst eine benötige Plattform für Smart Contracts ausgewählt und deren Komponenten analysiert. Anschließend wird auf die Entwicklungsumgebung und -werkzeuge für Smart Contracts eingegangen. Abschließend werden die Anforderungen, Herausforderungen und die Umsetzung des Prototyps beschrieben.

7.1 Auswahl von Smart Contract und DAPP-Plattform

Im Gegensatz zu reinen Kryptowährung-Plattformen bieten Smart Contract bzw. DAPP-Plattformen durch die Ausführung von Programmcode und Algorithmen mehr Möglichkeiten, als nur den Transfer von virtuellem Geld. Nachfolgend werden die am Markt verfügbaren Plattformen analysiert und bewertet. Anschließend wird eine Plattform festgelegt, auf die der zuvor fachlich konzipierte Geschäftsvorfall implementiert wird.

7.1.1 Plattformen

Anders als die Thematik um Kryptowährungen sind Plattformen für Smart Contracts trotz der zeitlich früheren Idee als Bitcoin, noch ein recht neues Thema. Die Plattformen befinden sich selbst noch in einem sehr frühen Stadium, womit mit Fehlern und wenig optimierten Umgebungen bzw. Workflows zu rechnen ist.

Ethereum

Das Projekt Ethereum ist eine von Vitalik Buterin 2013 entwickelte und 2014 von Gavin Wood formal beschriebene Smart Contract bzw. DAPP-Plattform mit einer eigenen „Turing-Vollständigen“[10] Programmiersprache (EthereumC, 2015), (Wood Dr., 2014). Zum gegenwärtigen Zeitpunkt befindet sich die Plattform noch in der Entwicklungsphase. Ein erstes Release hat es Mitte 2015 mit „Frontier“ gegeben. Anders als bei der Bitcoin-Plattform ist es möglich, ausführbaren Code ebenfalls durch Transaktionen in die Blockchain zu injizieren, welcher von den Knoten im Netzwerk aufgerufen werden kann. Diese Programme werden durch eine Aktivierung mittels einer weiteren Transaktion von jedem beteiligen Knoten des Netzwerkes in einer auf dem Knoten befindlichen Blockchain-basierten virtuellen Maschine, der EVM, ausgeführt. Darüber hinaus sind in „Ethereum“ auch finanzielle Transaktionen mit der eigenen Währung „Ether“ möglich. Daraus folgt, dass auf dieser Plattform ebenfalls

[10] Turing-Complete bedeutet, dass der Algorithmus durch eine Turing-Maschine, z.B. ein Automat, ausgeführt werden kann, http://c2.com/cgi/wiki?TuringComplete

finanzielle Anwendungsfälle realisiert werden können. Weiteres zu diesem Konzept und den Komponenten ist im Kapitel 7.1.3 beschrieben.

Eris Industries

Im Gegensatz zu „Ethereum“, verfolgt „Eris Industries“ den Ansatz, dass zu jeder DAPP eine eigene, geschlossene Blockchain genutzt wird und nicht wie bei „Ethereum“, eine Blockchain mit einem ausreichend großen Netzwerk an Knoten zur Validierung dahintersteht. Die gesicherte Validierung der Transaktionen erfolgt bei „Eris Industries“ indirekt dadurch, dass über eine Zugriffsschicht nur Berechtigten die Interaktion mit der Blockchain gestattet wird (Eris Industries, 2016). Durch dieses Konzept wird die Blockchain-Technologie in ihrer reinen Form als verteilte Datenbank mit Validierung und Speicherung von Transaktionen genutzt.

Codius

Das Projekt „Codius“ ist eine ehemalige Plattform für das Hosting von DAPPs. Das Projekt wird seit Mitte 2015 von der Firma Ripple Labs nicht weiterentwickelt (Maxim, 2015). Das Konzept beinhaltet unter anderem die explizite Einbeziehung von Smart Oracles, also der Bezug von externen Daten in der Blockchain und der Zugriff auf Dienste der Außenwelt. Die Abbildung 23 veranschaulicht dieses Konzept.

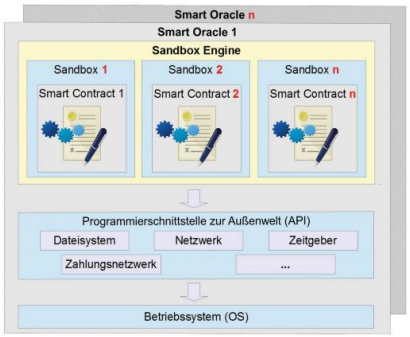

Abbildung 23: Konzept von Smart Oracles nach Codius[11]

Bitcoin

Die bekannteste Plattform für Kryptowährung ist zur Implementierung und Nutzung von Smart Contracts nicht ausgelegt. Zwar bietet Bitcoin ebenfalls eine Skriptsprache, welche aber im Gegensatz zu „Ethereum“ Einschränkungen bezüglich der Ausführung und Nutzungsmöglichkeiten mit sich bringt. Es sind keine Prüfungen oder die Ausführung von zusätzlicher Logik möglich (Bitcoinwiki-M, 2016). Eine weitere Einschränkung ist, dass die

[11] Entnommen aus Plattformsicherheit - Smart Contracts und TPM, Abbildung 1.1 von Christof Graff, Matthias Zscherp, Helmut Stoiber,entwickler.press, 2015

Block Size auf nur einem Megabyte gesetzt ist, was die Speicherung von internen Werten und Quellcode eines Smart Contracts einschränkt (Bitcoinwiki-N, 2016).

7.1.2 Auswahl der Smart Contract-Plattform

Als Smart Contract bzw. DAPP-Plattform ist das Projekt „Ethereum“ ausgewählt worden. Zum einen durch die Vielzahl an Projekten und Start-Ups sowie der Größe der Community, zum anderen soll in der vorliegenden Arbeit der Fokus nicht auf eine geschlossene Blockchain wie bei „Eris Industries“ gelegt werden, sondern auf eine berechtigungslose Blockchain bei der jeder mitwirken kann. Außerdem sind die Plattformen „Codius“ trotz des nützlichen Konzepts, aber aufgrund der fehlenden Weiterentwicklung, als auch die Bitcoin-Plattform, wegen der fehlenden Funktionalität, nicht weiter in Betracht gezogen worden.

7.1.3 Aufbau und Komponenten von Ethereum

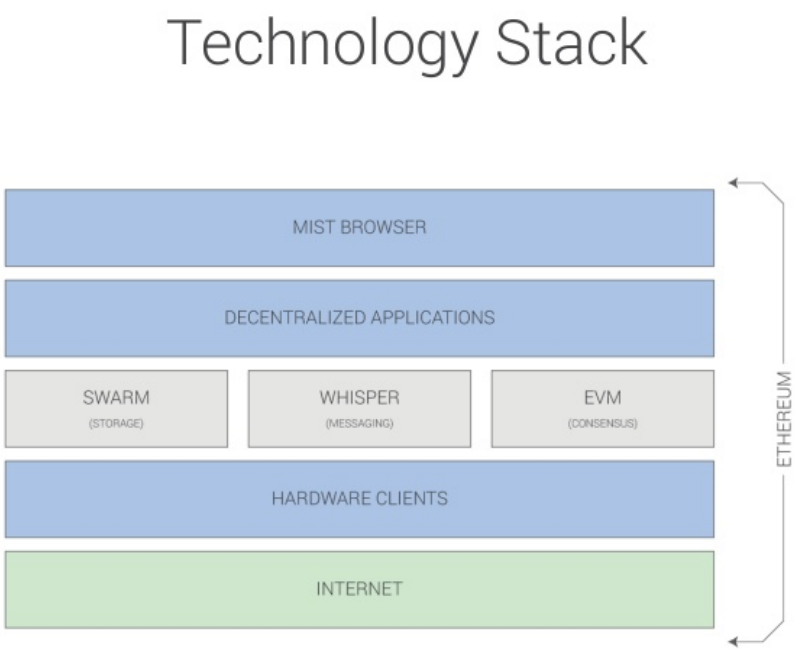

Abbildung 24: Ethereum Technologie Stack[12]

MIST Browser

Der MIST Browser ist eine sich in der Entwicklung befindliche grafische Oberfläche zur Benutzung von DAPPs. Daneben bietet er die Funktion einer Wallet zur Verwaltung der

[12]http://image.slidesharecdn.com/ethereum-stephantual-pres-8-150310095125-conversion-gate01/95/ethereum-paris-w-stephan-tual-17-638.jpg?cb=1425981727

Währung „Ether“ an. Das Programm basiert auf der Meteor-Plattform und wird als eigenständige, dezentrale Anwendung von einem Benutzer ausgeführt (EthereumE, 2015).

Ethereum Virtual Machine - EVM

Die Ethereum Virtual Machine, kurz EVM, ist eine verteilte virtuelle Maschine, die im Client auf jeden Knoten des Ethereum-Netzwerk betrieben wird. Diese dient sowohl der Ausführung eines kompilierten Smart Contracts, als auch der Verwaltung eines internen Zustands von Objekten sowie einer Kopie der Blockchain auf einem Knoten. Dabei ist zu erwähnen, dass die EVM eine „Turing-Vollständige“ Programmiersprache besitzt, die dazu dient, Smart Contracts auszuführen, um den Zustand der EVM, als auch von Objekten zu ändern. Wird eine Funktion eines Smart Contracts aufgerufen, so hat dies zur Folge, dass alle Knoten des Ethereum-Netzwerks diese ausführen. Bei größeren Operationen kann dies unter Umständen zu einem Engpass und so zu einem Effizienzproblem führen. Daher ist es zum gegenwärtigen Zeitpunkt empfehlenswert, nur kleine Operationen in einem Smart Contract auszuführen (EthereumD, 2015).

Clients

Zur Interaktion mit dem Netzwerk stehen dem Benutzer verschiedene Clients in verschiedenen Programmiersprachen zur Verfügung. Die Funktionsweise und die bereitgestellten Funktionen sind gleich. Die meisten Implementierungen sind als Konsolen-Anwendungen realisiert.

Das Programm „Eth“ ist ein in C++ geschriebener Client, welcher ebenfalls als Grundlage für den Client „AlethZero“ dient. „AlethZero“ bietet, anders als „Eth“, eine grafische Benutzeroberfläche zur Interaktion mit dem Ethereum-Netzwerk an.

Daneben gibt es den in Python implementierten Client „PyEthereum“. Auch dieser Client stellt wie „Eth“ die gleichen Funktionen, wie z.B. eine JSON-RPC-Schnittstelle für DAPPs zur Interaktion mit „Ethereum“, bereit.

Als letztes sei der Client „Geth“ erwähnt. Dieser Client ist in Golang realisiert und ist eine leichtgewichtige Anwendung zur Interaktion mit dem Ethereum-Netzwerk. Eine weitere vorteilhafte Eigenschaft von „Geth“ ist die interne Nutzung der JavaScript API „Web3“ innerhalb der „Geth“-Konsole (EthereumD, 2015). Daraus ergeben sich Synergieeffekte bei der Entwicklung der DAPP, welche ebenfalls auf dieser API zur Interaktion mit dem Ethereum-Netzwerk aufbaut. Aus diesem Grund wird der „Geth“-Client in der Version 1.4.0-stable als JSON-RPC-Schnittstelle und Übermittler der Befehle aus der DAPP zur Kommunikation mit „Ethereum“ genutzt.

Die in Abbildung 24 zusätzlich dargestellten Komponenten Swarm[13], ein verteiltes Dateisystem zur Speicherung von Daten und Whisper[14], das interne Kommunikationsprotokoll zum Austausch von Informationen zwischen Smart Contracts bzw. DAPPs, werden aufgrund der geringen Relevanz für die prototypische Implementierung des Geschäftsvorfalls nicht näher betrachtet.

7.1.4 Interkation mit dem Ethereum-Netzwerk und dem Smart Contract

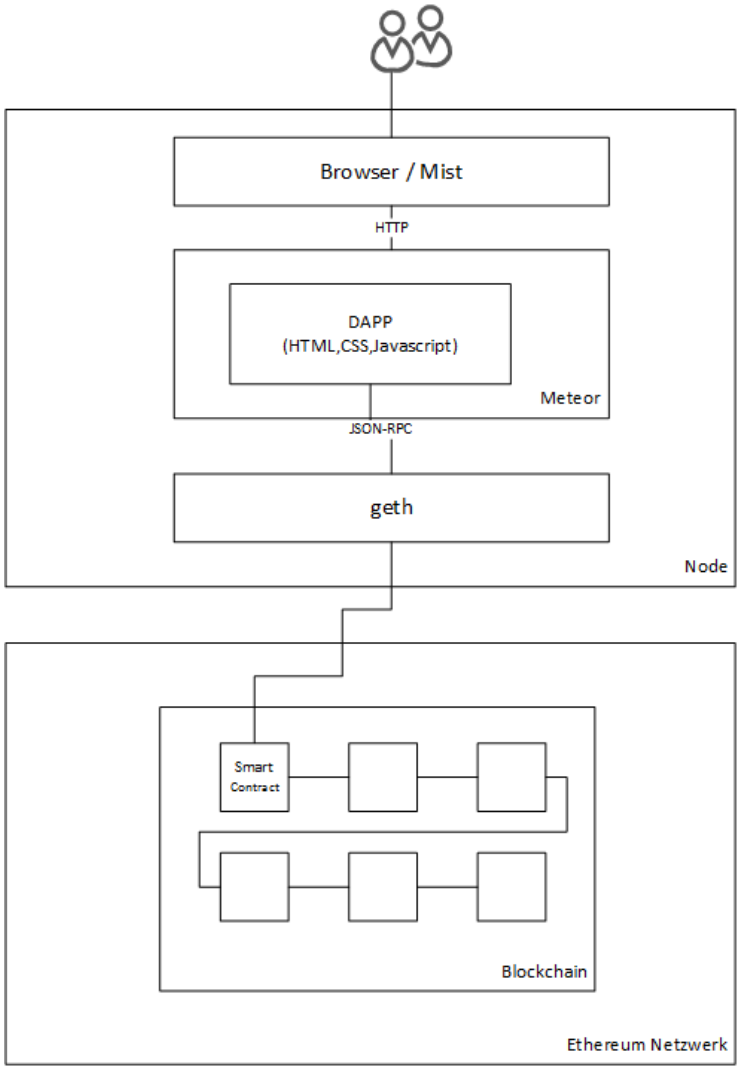

Abbildung 25: Interkation mittels DAPP und Ethereum-Netzwerk

Ein Benutzer kann mit dem Ethereum-Netzwerk und damit auch der Smart Contract über die zuvor beschriebenen Clients oder einer DAPP kommunizieren. Zukünftig soll es ebenfalls möglich sein, dass die Interaktion mit Instanzen von Smart Contracts mit einem eigens dafür

[13] Swarm, https://github.com/ethereum/go-ethereum/wiki/Swarm---distributed-preimage-archive

[14] Whisper, https://github.com/ethereum/wiki/wiki/Whisper

entwickelten Browser, dem MIST Browser, erfolgen kann. Zum gegenwärtigen Zeitpunkt befindet sich dieser noch in der Entwicklung (siehe Kapitel 7.1.3, MIST Browser).

Im Gegensatz zum Entwickler, ist die Idee für den Nutzer, dass dieser über einen Browser mittels des HTTP-Protokolls auf eine DAPP zugreifen kann. Diese repräsentiert dabei das Frontend und bildet die Schnittstelle zur Interaktion mit dem dahinterliegenden Smart Contract, welcher das Backend repräsentiert. Im Gegensatz zu nativen Clients, wie z.B. „Eth“ oder „Geth“ ist eine direkte Kommunikation mit dem Ethereum-Netzwerk nicht möglich. Als Kommunikationsschnittstelle muss daher im Hintergrund ein entsprechender nativer Client gestartet werden. Dieser wird durch die DAPP mittels der JavaScript-API „Web3“, welche von Ethereum entwickelt wird, durch JSON-RPC-Aufrufen angesprochen. Dadurch werden die Befehle von der DAPP durch den nativen Client an das Ethereum-Netzwerk gesendet und Rückmeldungen durch Callback-Methoden entgegengenommen. Durch die Verbindung zwischen dem Client und der JavaScript-API ist es möglich, sämtliche Funktionalitäten, die mit dem nativen Client möglich sind auch über die DAPP auszuführen.

7.2 Werkzeuge und Plattformen

7.2.1 Programmiersprachen

Sowohl für den Smart Contract, als auch für die DAPP werden unterschiedliche Programmiersprachen bzw. Mark-Up-Sprachen verwendet. Die Sprachen für das Frontend stammen aus dem Bereich der Web-Entwicklung.

Smart Contract

Ein Smart Contract kann in verschiedenen Programmiersprachen implementiert werden. Unter anderem ist es möglich, diese in Serpant, LLC, Mutan oder Solidity zu schreiben. Allen Programmiersprachen ist gemein, dass der Quellcode zu einem Kompilat übersetzt werden muss und nicht wie bei Skriptsprachen interpretiert wird, obwohl die jeweiligen Sprachen der Syntax von bekannten Skriptsprachen ähnlich sind.

Die in „Ethereum“ am meisten angewandte Programmiersprache zur Erstellung von Smart Contracts ist „Solidity“. Daneben gibt es in diesem noch recht jungen Umfeld Beispiel - Kontrakte in der Sprache „Serpant“. Aufgrund der zahlreichen Beispiele und Dokumentationen der Ethereum-Community wird „Solidity“ zur Implementierung des Smart Contracts für den entwickelten Geschäftsvorfall verwendet.

Solidity ist wie Java eine High Level - Programmiersprache. Zum gegenwärtigen Zeitpunkt befindet sie sich in der Entwicklung und liegt in der Version 0.20 vor (EthereumH, 2015). Aufgrund dessen sind Style Guides und Entwicklung - Patterns einem starken Wandel ausgesetzt. Sie ist stark an der Syntax von JavaScript orientiert, wodurch sich bei der

Entwicklung der DAPP und des Smart Contracts Synergieeffekte nutzen lassen. Dies ist ebenfalls ein ausschlaggebendes Kriterium, um „Solidity“ zur Implementierung des Smart Contracts zu verwenden.

DAPP

Eine dezentralisierte Applikation ist, wie bereits in Kapitel 2.2 erwähnt, eine auf Web-Technologien basierende Anwendung. Aus diesem Grund werden die zur Entwicklung von Web-Anwendungen überwiegend genutzten Programmier- und Mark-Up Sprachen verwendet. Dazu zählen für die Struktur der Anwendung HTML5, für das Design CSS und für ein dynamisches Verhalten und zur Interaktion JavaScript (Johnston, et al., 2014).

7.2.2 Entwicklungswerkzeuge

Entwicklungsumgebungen und Editoren

Für die Entwicklung von Smart Contracts bzw. DAPPs für Ethereum können verschiedene speziell dafür entwickelte Werkzeuge wie „Mix“, aber auch aus der Web-Entwicklung bekannte erweiterte Texteditoren, wie z.B. „Notepad++“ oder „Sublime“, verwendet werden.

Die Entwicklungsumgebung „Mix“ ist eine sich in der Entwicklung befindliche IDE aus dem Projekt Ethereum. Neben den bekannten Entwicklungsfunktionen bietet sie spezielle Funktionen für das Debugging von Smart Contracts an. Dazu gehört das Erstellen von Szenarien, welche eine Test-Blockchain in einen bestimmten Zustand zur Interaktion mit dem Smart Contract versetzt. Dabei verwendet „Mix“ den Client „Eth“ als Grundlage zur Interaktion mit der Blockchain (EthereumF, 2015).

Daneben ist es möglich, die Entwicklung und die Programmierung eines Smart Contracts bzw. einer DAPP in „Notepad++“ bzw. „Sublime“ vorzunehmen. Im Gegensatz zu Mix muss dabei das Deployment und Debugging manuell über einen der Clients erfolgen. Zur Unterstützung dieser Vorgehensweise sind ebenfalls Entwicklungs-Frameworks verfügbar, welche diese Prozesse erleichtern. Diese Frameworks werden in Kapitel 7.2.3 evaluiert.

Aufgrund der einschlägigen Erfahrung mit „Sublime“, der erhöhten Einarbeitungszeit bei der Verwendung sowie dem Entwicklungsstatus von „Mix“ wird zur Entwicklung des Smart Contracts bzw. der DAPP „Sublime“ (Build 3083) in Kombination mit dem „Geth“-Client verwendet.

7.2.3 Frameworks

Zur Unterstützung der Entwicklung der DAPP werden die Frameworks „Meteor“, „Embark“ und „Truffle“ betrachtet. Hier ist anzumerken, dass „Meteor“ mehr Funktionalität bietet als nur für eine DAPP für „Ethereum“. Dieses Framework zur Entwicklung von auf JavaScript-basierenden Web-Anwendungen ist generisch gehalten und wird für dezentralisierte Applikation aller Art verwendet. Im Gegensatz dazu sind die Frameworks „Embark“ und „Truffle“ speziell für die Entwicklung von DAPPs für „Ethereum“ entwickelt worden. Beide Frameworks sind sich aufgrund dessen ähnlich.

Bewertung der Frameworks

Die Frameworks sind unter Beachtung der für die Entwicklung relevanten Kriterien näher analysiert worden. Die Ergebnisse der Analyse werden im Anhang K dargestellt. Die dort abgebildete Matrix bildet die Grundlage für die Auswahl eines Frameworks für die Entwicklungs- und Laufzeitumgebung.

Auswahl von Frameworks

Aufgrund der teilweise engen Verzahnung von „Embark“ mit „Meteor“ werden beide Frameworks bzw. Applikationsplattformen für die DAPP genutzt. Auf der einen Seite wird das erleichterte und automatisierte Deployment von Smart Contracts durch „Embark“ genutzt, auf der anderen Seite die Erweiterbarkeit der webbasierten DAPP durch eine große Anzahl an verfügbaren Meteor-Paketen. Dabei ist zu erwähnen, dass „Embark“ ausschließlich zum initialen Aufsetzen der DAPP genutzt wird. Das erneute Deployment eines Smart Contracts nach Änderung wird manuell über den „Geth“-Client vollzogen. Dies ist damit begründet, dass es in der vorliegenden Entwicklungsumgebung zu einem fehlerhaften Auslesen der Parameter für den Smart Contract in „Embark“ gekommen ist. Somit ist ein automatisiertes Deployment nicht möglich gewesen.

7.2.4 Laufzeitumgebung

Die Laufzeitumgebung der DAPP basiert auf „Meteor“ (Version 1.2.1). Dies ist wie bereits erwähnt ein Framework zur Entwicklung von JavaScript-basierten Web-Anwendungen. Aus diesem Grund nutzt Meteor zur Ausführung der DAPPs im Hintergrund Node.js. Node.js ist eine serverseitige Anwendungsplattform zum Betreiben von Web-Anwendungen, welche in JavaScript geschrieben worden ist. Dadurch ist es möglich, leichtgewichtige Webserver und

damit ebenfalls DAPPs zu realisieren. Als ausführende JavaScript-Engine wird die von Google entwickelte Laufzeitumgebung „V8“ verwendet[15].

7.3 Prototypische Umsetzung des Smart Oracles

Für die protypische Umsetzung des in Kapitel 5.2 gewählten Geschäftsvorfalls ist in Kapitel 7.1.2 ein Smart Contract in der Ethereum Blockchain gewählt worden. Für die Umsetzung des Geschäftsvorfalls in einen in der Ethereum Blockchain gehosteten Smart Contract, wird ein weiterer Prototyp benötigt. Der Prototyp eines Smart Oracles, um den Smart Contract mit Marktdaten versorgen zu können.

Die Entscheidung einen eigenen Prototyp zu entwickeln und nicht auf eine der in Kapitel 6.2 genannten Lösungen zurückzugreifen ist darin begründet, dass sich die vorgestellten Lösungen primär an die Bitcoin Blockchain richten oder nur eine Integration in die öffentliche Ethereum-Blockchain und nicht in eine private Ethereum-Test-Blockchain anbieten. Daneben bietet eine eigene Implementation mehr Flexibilität, insbesondere im Hinblick auf die externe Datenversorgung und das Testen der Implementierung.

7.3.1 Funktionale Anforderung

Die hier genannten funktionalen Anforderungen und im folgenden Unterkapitel genannten nicht-funktionalen Anforderungen sind aus den Bedürfnissen des Smart Contracts an die Marktdatenversorgung und den Möglichkeiten sowie Einschränkungen der eingesetzten Technologie Ethereum abgeleitet worden.

Registrieren eines Smart Contracts

Der Smart Contract, welcher mit Marktdaten versorgt werden will, muss die Möglichkeit besitzen sich am Oracle Contract zu registrieren und die für die Marktdatenversorgung benötigten Ether zu überweisen.

Abmelden eines Smart Contracts

Ein mit Marktdaten versorgter Smart Contract muss die Möglichkeit haben, die Marktdatenversorgung durch den Smart Oracle Contract abzubestellen und die nicht verwendeten Ether zurückzuerhalten.

[15] https://nodejs.org/en/docs/es6/

Regelmäßige Marktdatenversorgung

Der registrierte Smart Contract muss in einem zu definierenden Intervall mit realen Marktdaten versorgt werden.

Abrechnung der Dienstleistung

Das Smart Oracle muss dem registrierten Contract marktgerechte Gebühren für die Bereitstellung der Marktdaten und das Aufrufen von Funktionen berechnen.

7.3.2 Nicht-funktionale Anforderungen

Modularer Aufbau

Der zu entwickelnde Prototyp soll nach Möglichkeit modular aufgebaut werden, um so eine einfache Wiederverwendbarkeit der Module in anderen Projekten zu ermöglichen. Daneben sollen die Datenquellen möglichst einfach austauschbar sein, um theoretisch eine möglichst große Anzahl an Marktdaten anbieten zu können und nicht an einen Anbieter gebunden zu sein.

Vertrauenswürdige Datenquellen

Die Marktdaten sollen von Institutionen bezogen werden, welchen allgemein ein großes Vertrauen entgegengebracht wird.

Robustheit

Etwaige Fehler im Prozess der Marktdatenversorgung soll der Prototyp nach Möglichkeit selber kompensieren. Nicht kompensierbare Fehler sollen auf der Konsole bzw. einer Log-Datei ausgegeben werden.

7.3.3 Konzeption des Smart Oracles

Die Konzeption des Smart Oracles unterteilt sich in zwei Bereiche. Zum einen muss das eigentliche Oracle als Java-Anwendung entwickelt werden. Zum anderen soll ein Oracle Contract entwickelt werden, welcher die neuen Marktdaten des eigentlichen Oracles entgegennimmt und an den Smart Contract weiterreicht. Dieser Zwischenschritt erhöht die Modularität der Gesamtanwendung, da das Oracle so nicht nur für den einen Anwendungsfall genutzt werden kann, sondern für mehrere unterschiedliche Smart Contracts.

Das nachfolgende BPMN-Diagramm soll den Prozessablauf des Smart Oracles und das Zusammenspiel zwischen den Entitäten verdeutlichen. Ein vergrößerte Darstellung befindet sich in Anhang P.

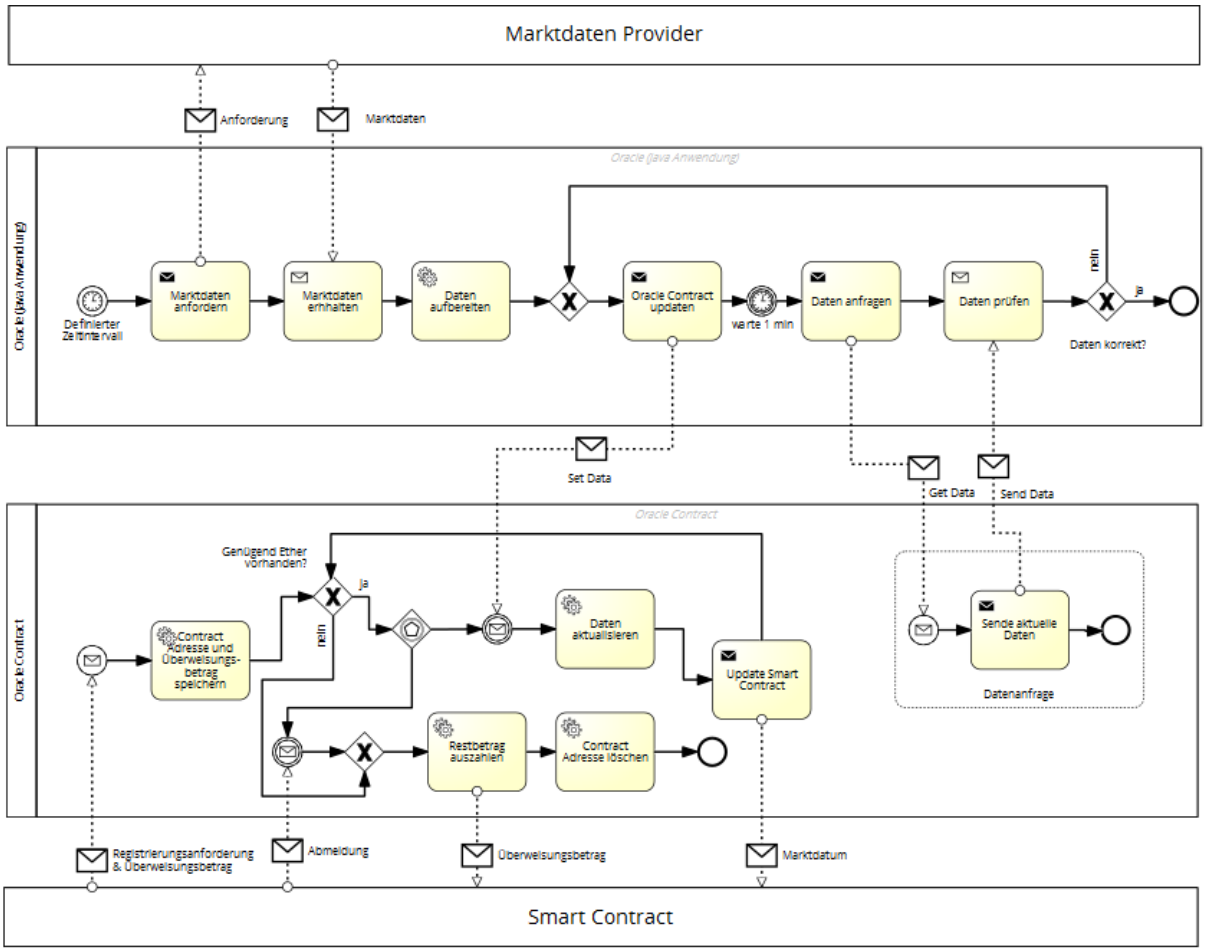

Abbildung 26: Prozessablauf Smart Oracle

Der dargestellte Prozess beginnt aus der Sicht des Oracles damit, dass ein Smart Contract sich am Oracle Contract registriert und Ether für die Dienstleistung des Oracles überweist. Der Oracle Contract registriert daraufhin den Contract und verbucht die erhaltenen Ether intern. Nach der Prüfung, ob genügend Ether vorhanden sind, wartet der Oracle Contract darauf, ob neue Marktdaten von der Java-Anwendung bereitgestellt werden oder sich der registrierte Smart Contract wieder abmeldet. Kommt es aufgrund von einer selbst gewählten Abmeldung des Contracts oder aufgrund von nicht genügend vorhandener Ether zu einer Abmeldung des Contracts, so werden die verbleibenden Ether ausbezahlt und die Adresse des Smart Contracts wird gelöscht. Sind genügend Ether vorhanden und es treffen neue Marktdaten von der Java-Anwendung ein werden diese an den registrierten Smart Contract weitergereicht. Danach beginnt der Prozessabschnitt mit der Prüfung des Etherkontostandes erneut.

Der Prozess der Java-Anwendung wird in einem definierten Zeitintervall automatisch gestartet. Nach dem Start ruft die Anwendung externe Marktdaten von einem Dienstleister ab und bereitet diese für die Verwendung in einem Smart Contract auf. Danach erfolgt die eigentliche Übertragung der Marktdaten an den Oracle Contract. Nach einer Wartezeit von einer Minute erfolgt eine Überprüfung, ob die übertragenen Daten im Oracle Contract richtig

gesetzt wurden. Sollten die Daten nicht richtig übertragen worden sein, so wird das Setzen der Daten erneut angestoßen.

7.3.4 Eingesetzte Technologien

Für die Umsetzung des Smart Oracles werden die folgenden Technologien eingesetzt:

Java
Für die Umsetzung des eigentlichen Oracles soll die Programmiersprache Java in der Version 8 verwendet werden. Die Programmiersprache Java wird als bekannt vorausgesetzt und an dieser Stelle nicht weiter erläutert. Als Build-Management-Tool wird Apache Maven in der Version 3.3.3 eingesetzt.

Ethereum/Solidity
Für die Implementierung des Oracle Contracts wird die für Ethereum entwickelte Programmiersprache Solitity eingesetzt. Eine detaillierte Beschreibung zu Ethereum und der Programmiersprache Solidity ist in Kapitel 7.2 zu finden.

Geth
Für die Kommunikation mit der Ethereum Blockchain und somit auch dem Oracle Contract wird ein Geth-Client eingesetzt. Eine Beschreibung zu Geth befindet sich ebenfalls in Kapitel 7.2.

JSON-RPC
Die Kommunikation zwischen der Java-Anwendung und dem Geth-Client erfolgt über Remote Procedure Calls (RPC), welche vom Geth-Client bereitgestellt werden. Als Datenformat akzeptiert der Geth-Client das JSON-Datenformat.

7.3.5 Datenquelle

Als vertrauenswürdige Datenquelle wird die an allen Handelstagen bereitgestellten XML-Datei der Europäischen Zentralbank verwendet, welche alle bedeutenden Wechselkurse gegenüber dem Euro enthält (Europäische Zentralbank). Der modulare Aufbau soll aber einen Austausch der Datenquelle ermöglichen.

7.3.6 Umsetzung

Die Betrachtung der Umsetzung erfolgt in zwei Teilen. Zunächst wird der in Solidity geschriebene Oracle Contract betrachtet. Im zweiten Teil wird dann auf das eigentliche Oracle als Java-Anwendung eingegangen.

Oracle Contract

Aufgrund der geringen Größe des Oracle Contracts wird an dieser Stelle der Quellcode komplett dargestellt und erläutert. Eine Abstrahierung ist daher nicht notwendig.

```
contract oraclecontract{

 address oracleAddress = 0xae0209e12acf92cf1143b4e4ac0e9df7f8f178ac;
 address public registeredContract;
 uint public registeredContractBalance;
 clearinghouse public ch;
 uint public currentValue;
 uint public updateTime;

    function set(uint x) {
        ch = clearinghouse(registeredContract);
        if (msg.sender != oracleAddress)
            throw;
        currentValue = x;
        updateTime = now;
        if (registeredContractBalance >= 1000){
            ch.updateMarketPrice(x);
            oracleAddress.send(1000);
            registeredContractBalance= registeredContractBalance - 1000;
        }
    }
    function get() constant returns (uint retVal) {
        return currentValue;
    }
    function getUpdateTime() constant returns (uint retVal){
        return updateTime;
    }
    function register (){
        if (registeredContract != 0)
            throw;
        registeredContract = msg.sender;
        registeredContractBalance = msg.value;
    }
        function unregister (){
        if (msg.sender != registeredContract)
            throw;
        registeredContract.send(registeredContractBalance);
        registeredContract = 0;
        registeredContractBalance = 0;
    }
}
```

Abbildung 27: Quellcode Oracle Contract

Im obersten Abschnitt des Oracle Contracts sind die Variablendeklaration zu finden. Über die oracleAddress wird festgelegt, von welchem Oracle der Oracle Contract Marktdatenupdates entgegennimmt. In den Variablen registeredContract und registeredContractBalance werden später die Adresse und der Kontostand des registrierten Smart Contracts gespeichert. Die Variable ch vom Typ Clearinghouse ist notwendig, damit der Oracle Contract im späteren

Verlauf eine Funktion beim registrierten Contract aufrufen kann, um so das Update der Marktdaten durchzuführen. In der Variable currentValue wird der aktuelle Wert des Marktdatums festgehalten. UpdateTime speichert die zugehörige Aktualisierungszeit.

Die Hauptfunktionalität findet in der Funktion set statt. Sie wird von der Java-Anwendung aufgerufen und bekommt das aktuelle Marktdatum übergeben. Nach dem Aufrufen wird zunächst die Variable ch mit der Adresse des registrierten Contracts befüllt. Dies ist notwendig, damit im weiteren Verlauf die Funktion updateMarketPrice() am richtigen Smart Contract aufgerufen wird. Mit der if-Funktion wird sichergestellt, dass die Funktion nur vom zuvor definierten Oracle aufgerufen werden kann. Ist dies sichergestellt, so werden der aktuelle Wert und das Datum neu gesetzt. Daraufhin erfolgt, sofern der registrierte Contract über genügend Ether-Einlagen verfügt, das Aufrufen der updateMarketPrice()-Funktion beim registrierten Contract. An die Oracle Adresse werden die für die Transaktion berechnete Ether überwiesen, da das aufrufende Oracle für die gesamte Transaktionskette die Transaktionskosten bezahlen muss. In der letzten Zeile der Funktion wird der Kontostand des registrierten Contracts um den berechneten Wert reduziert.

Die Funktionen get() und getUpdateTime() sind Hilfsfunktionen um das aktuelle Marktdatum und die Aktualisierungszeit auslesen zu können.

Über die Funktion register() kann sich ein Smart Contract registrieren und gleichzeitig die für die Dienstleistung erforderlichen Ether überweisen. Die Funktion ist nur aufrufbar, wenn noch kein Smart Contract registriert ist.

Über die Funktion unregister() kann sich der registrierte Contract wieder von der Marktdatenversorgung abmelden und erhält seinen verbleibenden Kontostand zurückerstattet.

Java-Anwendung

Die Umsetzung der Java-Anwendung wird anhand des nachfolgenden Klassendiagrammes erläutert. Der vollständige Quellcode befindet sich als Anhang Q.

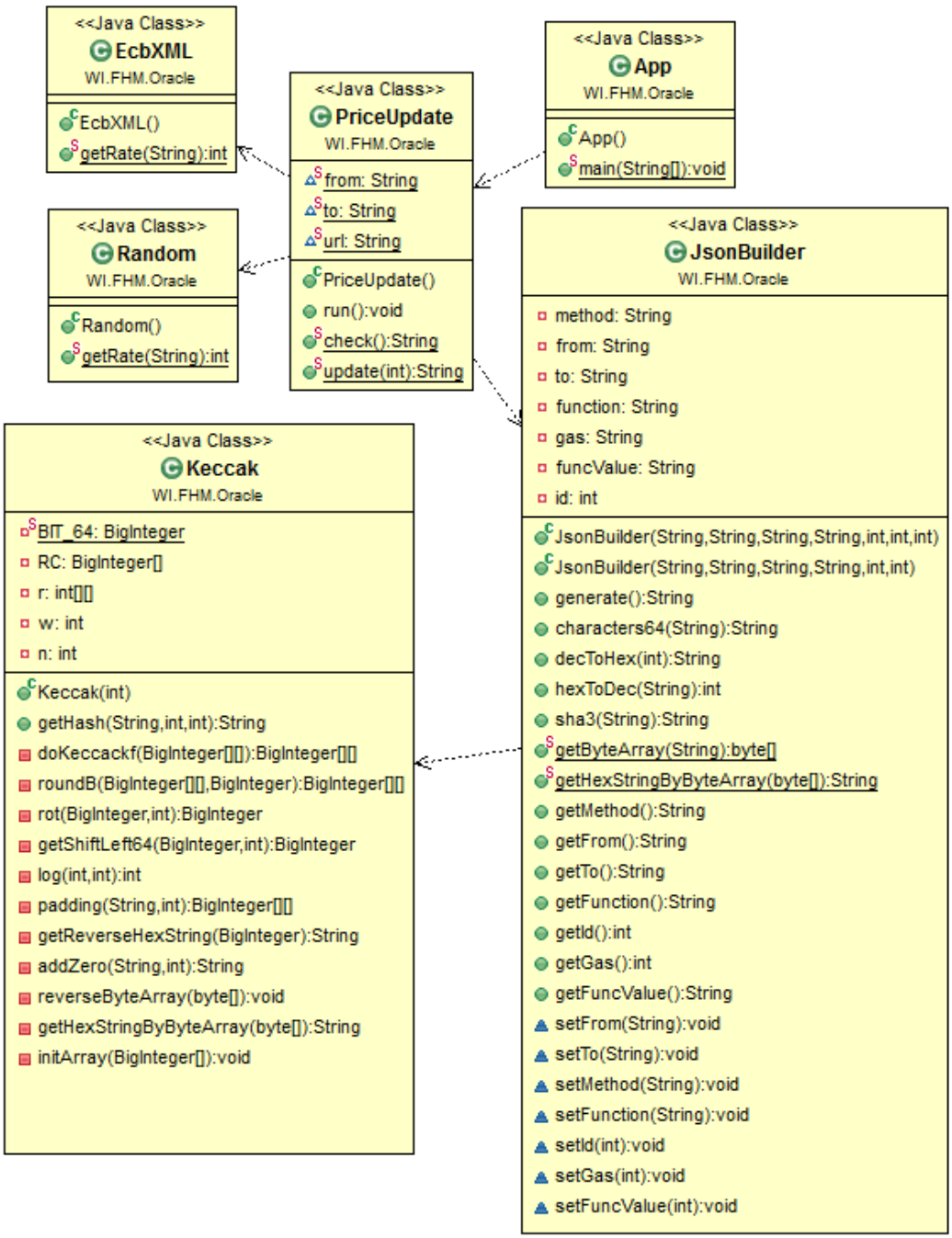

Abbildung 28: Klassendiagramm Oracle (Java-Anwendung)

Die Klasse App ist die ausführbare Klasse der Anwendung. Über einen Übergabeparameter kann der Intervall für die Marktdatenaktualisierung in Minuten festgelegt werden. In diesem Zeitintervall wird die Klasse PriceUpdate wiederholt aufgerufen. Aus einer Properties Datei liest die Klasse PriceUpdate die benötigten Parameter für eine Marktpreisaktualisierung. Dies sind: Die Adresse des absendenden Accounts, die Zieladresse des Oracle Contracts, die URL unter welcher der RPC-Server erreichbar ist und die Währung des benötigten Wechselkurses. Über die boolesche Variable *random* kann festgelegt werden, ob die Klasse PriceUpdate die Klasse ECBXML aufruft, um reale Marktdaten zu erhalten, oder die Klasse Random aufruft, um Zufallswerte zum Testen zu erhalten.

Vor dem eigentlichen Update des Oracale Contracts wird die Klassen JsonBuilder benötigt. Sie bekommt alle benötigten Parameter übergeben, um den JSON-String zu erstellen. Hierzu bildet sie mit Hilfe der Klasse Keccak[16] den SHA3 Wert vom aufzurufenden Funktionsnamen des Oracle Contracts und verwendet die ersten vier Byte des SHA3-Wertes für den JSON-String. Des Weiteren übersetzt sie die übergebenen Dezimalwerte in das für den JSON-String benötigten Hexadezimalformat und füllt den Hexadezimalwert mit Nullen auf 32 Byte auf. Alle weiteren benötigten Werte, wie die Methode, die Absenderadresse und die Zieladresse werden ebenfalls in den JSON-String eingefügt.

Die Klasse PriceUpdate baut daraufhin eine RPC-Verbindung zum Geth-Client auf und überträgt den JSON-String zum Geth-Client. Hierdurch wird das aktuelle Marktdatum an den Oracle Contract übergeben.

Nach einer Wartezeit von einer Minute wird durch den Aufruf der get()-Funktion beim Oracle Contract überprüft, ob das aktuelle Marktdatum erfolgreich übertragen wurde. Sollte das Marktdatum nicht erfolgreich übertragen worden sein, so wird der JSON-String erneut an den Geth-Client gesendet.

Die wichtigsten Zwischenschritte und Ergebnisse werden zur besseren Nachvollziehbarkeit auf der Konsole ausgegeben.

7.3.7 Kritische Reflexion und Ausblick

Die Grundfunktionalität eines Smart Oracles konnte mit diesem Prototyp erfolgreich implementiert werden. Externe Marktdaten können persistent in der Ethereum-Blockchain gespeichert werden und anderen Smart Contracts zur Verfügung gestellt werden. Die in den Kapiteln 7.3.1 und 7.3.2 definierten funktionalen und nicht-funktionalen Anforderungen konnten vollständig umgesetzt werden.

Ein erhöhtes Vertrauensniveau besitzt dieser Prototyp hingegen nicht, da keine der in Kapitel 6.2 vorgestellten Methoden zur Vergrößerung des Vertrauens eingesetzt wurden. Vor einem praktischen Einsatz sollten mehrere unabhängige Oracles entwickelt werden und/oder das von oraclize.it vorgestellte kryptografisches Bestätigungsverfahren eingesetzt werden, um das Vertrauen zu erhöhen.

Daneben sollte das Oracle zukünftig so weiterentwickelt werden, dass mehrere unterschiedliche Marktdaten gleichzeitig zur Verfügung gestellt werden können und ebenfalls

[16] Die Klasse Keccak ist nicht selber geschrieben und wurde vom Entwickler romus übernommen: https://github.com/romus/sha/blob/master/sha3/src/com/theromus/sha/Keccak.java

mehrere Smart Contracts sich an einem Oracle registrieren können, um mit Marktdaten versorgt zu werden.

7.4 Technische Konzeption des Clearinghouse-Smart Contracts

7.4.1 Nicht-funktionale Anforderungen

Robustheit

Die DAPP bzw. der Smart Contract soll bei fehlerhaften Eingaben oder Ausführung von Aktionen innerhalb der DAPP sauber terminieren und gegebenenfalls ausgeführte Transaktionen rückgängig machen. Dies ist besonders wichtig, wenn die Transaktionen den Transfer von „Ether“ als Bestandteil haben, wie z.B. bei der der Überweisung der initialen Margin.

Integrität

Die internen Daten des Smart Contracts sollen nur nachvollziehbar geändert werden können. Diese Anforderung ist zum Teil durch die Nutzung der Blockchain-Technologie eingehalten. Des Weiteren soll es nur den im Smart Contract fest geschriebenen, vertraulichen Quellen, dem Oracle, möglich sein, die Preise zu ändern.

Besonderheit von Schutz privater Informationen

Eine Besonderheit bei der Nutzung der Blockchain-Technologie ist, dass wie bereits in Kapitel 3.3.1 erwähnt, die Daten für jeden Nutzer im Netzwerk öffentlich sichtbar sind. Daher ist in Bezug auf den Datenschutz bei einem solchen Vertrage diese Anforderung zu verwerfen, da die unterliegende Technologie dies ausschließt.

Benutzerfreundlichkeit

Die DAPP soll einfach zu bedienen sein, das heißt, ausführbare Funktionen des Smart Contracts sollen sofort ersichtlich und intuitiv bedienbar sein.

7.4.2 Funktionale Anforderungen

Die nachfolgend beschriebenen, funktionalen Anforderungen ergeben sich aus dem im Kapitel 5.2 fachlich konzipierten Geschäftsvorfall „Smart Contracts ersetzen Clearinghouse im Derivatehandel“. Eine Visualisierung des vollständigen, verzweigten Ablaufs befindet sich im Anhang L.

Aufsetzen von Kontrakt

Der grundlegende Smart Contract ohne jegliche Parameter ist in der DAPP fest verankert. Zusätzlich soll es über die DAPP möglich sein, Smart Contracts mit selbst definierter Parametern aufzusetzen.

Einzahlung von Margin

Eine grundlegende Funktion ist die Einzahlung der initialen Margin, um den Vertrag aktivieren zu können. Diese ist gleich oder höher als ein im Smart Contract definierten Prozentsatz vom eigentlichen Vertragsvolumen. Der Smart Contract soll erst aktiviert werden, wenn von beiden Vertragsparteien die Margin eingezahlt worden ist. Erst ab diesem Zeitpunkt beginnt die Laufzeit.

Margin Call

Eine zweite grundlegende Funktion ist der sogenannte „Margin Call“. Dieser Nachschuss von weiterem Geld durch eine Partei muss erfolgen, wenn der Kontostand einer Partei unter eine zuvor definierten „Maintenance Margin“ fällt. Die Höhe des zu leistenden Nachschusses ist dabei die Differenz zwischen aktuellem Kontostand und der anfangs eingezahlten Margin.

Auflösung von Kontrakt

Eine weitere wichtige Funktion ist die des Settlements. Nach Ablauf der Laufzeit des Smart Contracts oder der Naschschuss-Frist sollen „Ether“ der im Smart Contract gespeicherten Kontostände an die Vertragsparteien ausgezahlt werden. Dies entspricht der Funktion eines Clearinghouses (siehe Kapitel 5.1).

Einspielen von externen Marktdaten

Den Vertragsparteien müssen transparent aktuelle Marktdaten in der DAPP angezeigt werden können. Hierzu gehören der aktuelle Preis der Währung, die abgesichert werden soll und der Zeitstempel der letzten Aktualisierung.

Abruf von Kontraktdaten

Den Vertragsparteien sollen alle Informationen übersichtlich über den Vertrag in der DAPP dargestellt werden. Dazu zählen neben den Marktdaten, die aktuellen Angaben zu den Kontoständen und ob ein Margin Call erfolgen muss.

Abruf von Teilnehmerdaten

In der DAPP sollen ebenfalls die Account-Daten der einzelnen Vertragsparteien transparent und übersichtlich angezeigt werden. Hierzu zählen der Public Key, also die Adresse der jeweiligen Vertragsparteien in Ethereum, sowie die Position die sie im Kontrakt einnimmt.

7.4.3 Herausforderungen

Informieren über Frist von Nachschuss von Margin

Aufgrund des Umstandes, dass Smart Contracts nicht über den gesamten Zeitraum aktiv im Ethereum-Netzwerk sind, sondern nur, wenn sie von außen angestoßen werden und anschließend die entsprechenden Operationen ausführen, ergeben sich Einschränkungen bei der Einhaltung von Fristen. Hier besteht die Herausforderungen darin, den Vertragsparteien mitzuteilen, dass z.B. nach einer Aktualisierung des Preises ein Margin Call erfolgen muss. Ein Lösungsansatz hierfür ist, dass die Vertragsparteien in einem bestimmten Intervall die DAPP aufrufen müssen (z.B. einmal am Tag). Dadurch wird der Vertrag angestoßen und die Information über ein Margin Call zu den Vertragsparteien übermittelt.

Fristen prüfen und durchsetzen

Wie zuvor angesprochen, sind Smart Contracts reaktive Teilnehmer im Ethereum-Netzwerk. Dadurch ergibt sich eine weitere Herausforderung, weil nur über Umwege Fristen überprüft bzw. durchgesetzt werden können. Das Settlement eines Smart Contracts nach Ablauf der Laufzeit kann z.B. nur dann erfolgen, wenn einer der Vertragsparteien oder eine dritte unabhängige Partei die jeweilige Funktion des Smart Contracts über eine Transaktion anstößt. Sonst ist dies nicht möglich. Gleiches gilt für die Einhaltung der Frist für den Margin Call. Hier kann es unter Umständen zu verspäteten Mitteilungen vom Smart Contract zur jeweiligen Vertragspartei kommen, sodass diese Partei gar nicht die Möglichkeit hat, die Frist einzuhalten und der Vertrag somit aufgelöst worden ist. Ein Lösungsansatz ist hier, dass der Aufruf ebenfalls in einem bestimmten Intervall durch die jeweiligen Vertragsparteien erfolgen muss.

Bezahlung von Transaktionskosten für Marktdaten

Die Aktualisierung des Preises im Smart Contract durch das Oracle ist mit Transaktionskosten verbunden. Auch die Ausführung der Operation zur Aktualisierung muss bezahlt werden. Hierbei stellt sich die Frage, wie der Ablauf erfolgen muss, dass diese Kosten von den Vertragsparteien übernommen und bezahlt werden können. Ein Lösungsansatz ist, dass die Vertragsparteien beim Aufsetzen des Vertrages eine bestimmte Summe an „Ether“ zusätzlich zu der initialen Margin einzahlen müssen. Diese Summe wird anschließend vom Smart Contract bei der Registrierung am Oracle zum Erhalt von Aktualisierungen überwiesen. Eine Aktualisierung erfolgt nur solange, wie „Ether“ zur Bezahlung der Transaktionskosten vorhanden ist. Sollte die Laufzeit länger sein als die eingezahlte Summe, so ist ein Nachschuss von „Ether“ nötig oder es erfolgen keine Aktualisierungen mehr für den im Oracle-Smart Contract registrierten Smart Contract.

7.4.4 Konzeption des Clearinghouse-Smart Contracts

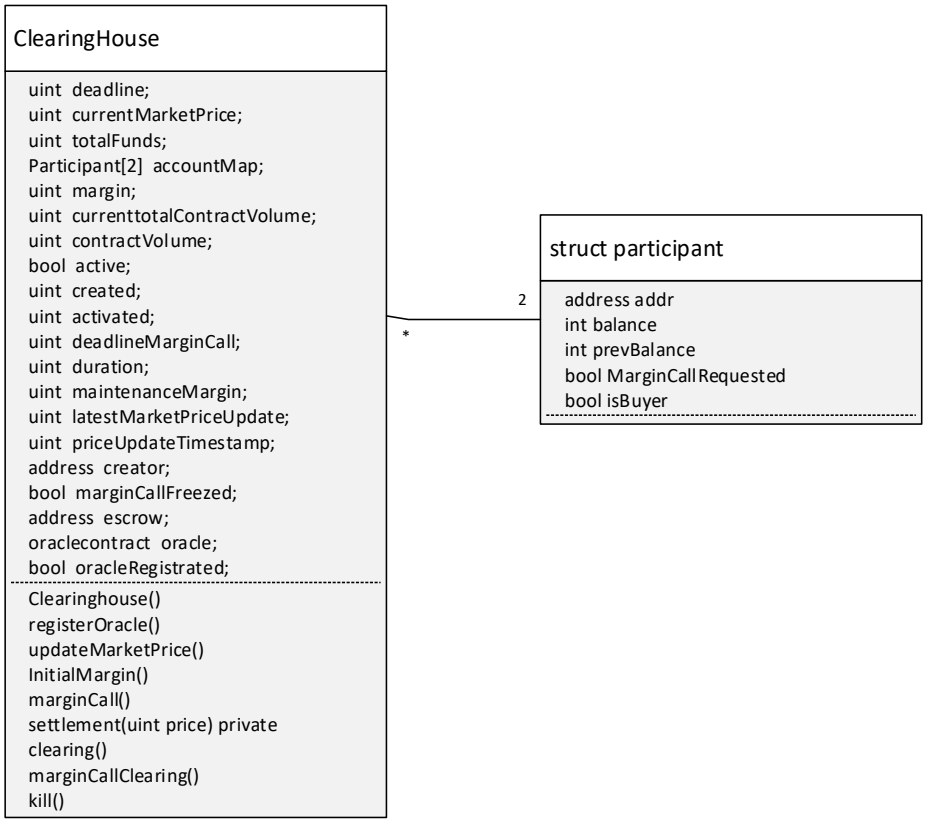

Abbildung 29: Klassendiagramme Clearinghouse Smart Contract

Der Clearinghouse-Smart Contract ist in zwei Klassen unterteilt. Anders als sonst üblich, erbt die Klasse Clearinghouse nicht von einem „mortal“-Kontrakt, welcher generalisierte Funktionen zum Abbau einer Smart Contract-Instanz enthält. Dies ist damit begründet, dass zusätzliche Logik für den Abbau einer Instanz notwendig ist (siehe Kapitel 7.5.5).

Des Weiteren werden Daten der Vertragsparteien separat in einem struct-Objekt gespeichert, das ausschließlich die für den Kontrakt dynamische, spezifische Daten enthält. Die Speicherung der Adresse der jeweiligen Vertragspartei ist hierbei notwendig, um eine eindeutige Zuordnung zu gewährleisten.

Die Variablen im Smart Contract sind mit dem Modifier „public“ deklariert. Daraus ergibt sich, dass diese über nicht explizit deklarierte Getter-Methoden abgerufen werden können. Die Setter-Methoden müssen dennoch explizit implementiert werden, da dafür eine Transaktion im Ethereum-Netzwerk benötigt wird, welche nach Absetzen in der Blockchain von den „Minern“ bestätigt werden muss. Ein Beispiel dafür ist die Aktualisierung des Marktpreises. Die Funktion „Clearing“ ist nur intern im Smart Contract aufrufbar. Dies ist damit begründet, weil ausschließlich die Daten der Vertragsparteien in dessen Datenstruktur nach Aktualisierung des Marktpreises berechnet werden. Alle weiteren Funktionen sind über Transaktionen im Ethereum-Netzwerk öffentlich ausführbar.

7.4.5 Konzeption der dezentralen Applikation

Softwarearchitektur

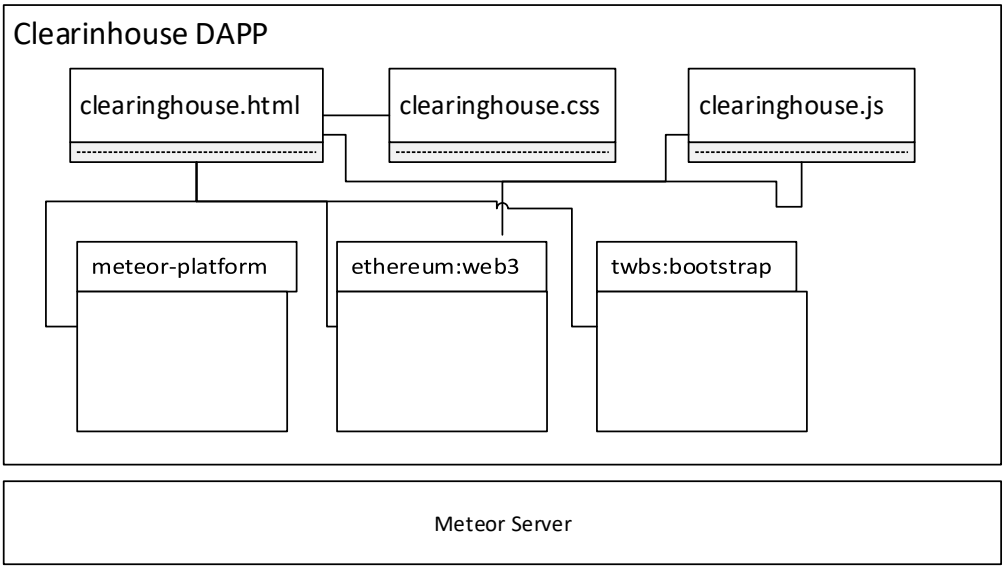

Abbildung 30: Software-Architektur der Clearinghouse DAPP

Der Aufbau der DAPP, für den zugrundeliegenden Smart Contract des Geschäftsvorfalls, folgt dem durch den von „Embark“ mit „Meteor“ vorgegebenen Aufbau. Dies hat zur Folge, dass aufgrund der geringen Komplexität drei Dateien mit unterschiedlichem Fokus auf einen Teil der DAPP vorhanden sind. Die Clearinghouse-HTML Datei implementiert die Grundstruktur der DAPP. Die Struktur der HTML-Seite ist für die Übersichtlichkeit nochmals in sogenannte Meteor-Templates unterteilt. Zusätzlich zu den für die Interaktion mit einem erstellten Smart Contract implementierten JavaScript-Funktionen, sind in der Clearinghouse JavaScript-Datei Helper- und Events- Methoden für die definierten Templates implementiert. So werden unter anderem durch Aufruf der Web3-JavaScript-Bibliothek von „Ethereum“ im Smart Contract gespeicherte Werte abgerufen. Die Clearinghouse CSS-Datei dient ausschließlich der Auslagerung von Design-Definition.

Die extern eingebundenen Meteor-Pakete dienen der einfachen Einbindung und Verwaltung von JavaScript bzw. CSS-Frameworks oder Bibliotheken. Hierzu zählen das Paket „twbs:bootstrap“, welches das von Twitter entwickelte JavaScript- und CSS-Framework „Bootstrap“ importiert und das Paket „ethereum:web3“, welches die JavaScript API für „Ethereum“ über das „Web3“-JavaScript-Objekt für den Zugriff auf Funktionalitäten von Ethereum bereitstellt. Das Paket „Meteor-Platform“ bildet die Grundlage für eine auf meteor-basierende DAPP und importiert Standard Meteor-Pakete in das Projekt.

DAPP-Oberfläche

Durch die Zweiteilung der Oberfläche der DAPP in einen Deployment-Teil sowie in einen Informationsteil wird die Benutzerfreundlichkeit und Übersichtlichkeit der Web-Anwendung gewahrt. Weiter ist wie Abbildung 32 zeigt, der Informationsteil in die drei Unterbereiche Markt-, Vertrags- und Account-Informationen unterteilt. Damit ist gewährleistet, dass alle relevanten Informationen für die Vertragsparteien sofort ermittelt werden können. Im Gegensatz dazu besteht der Deployment-Teil nur aus einer Eingabemaske für die Parameter des zu erstellenden Smart Contracts. Dieser Sachverhalt ist in Abbildung 31 veranschaulicht.

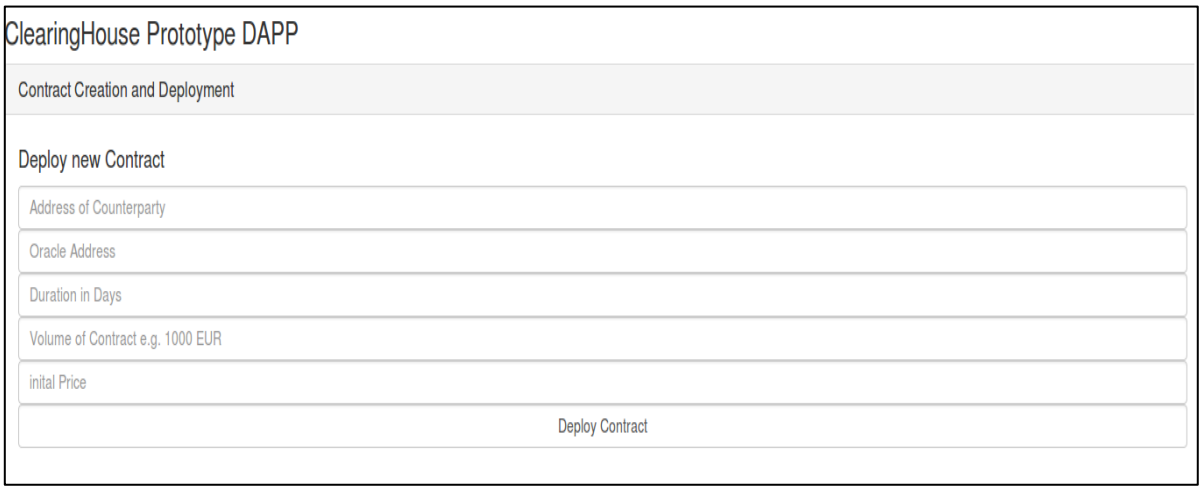

Abbildung 31: Deployment von Kontrakt

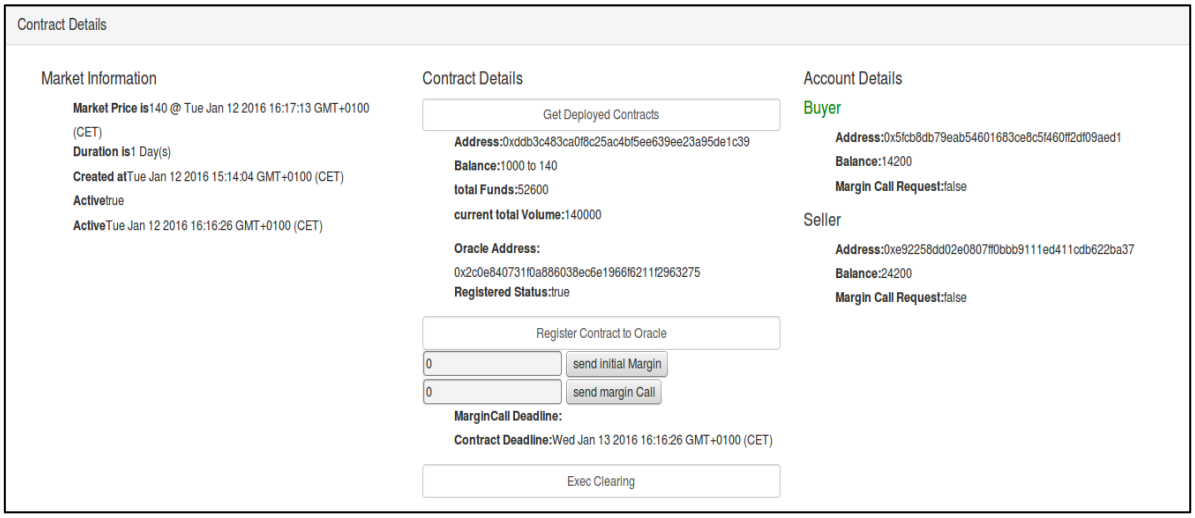

Abbildung 32: Kontrakt-Informationen

7.5 Umsetzung des Clearinghouse-DAPP-Prototyps

7.5.1 Oberfläche

Zur Gestaltung der Oberfläche für die DAPP wird das von Twitter entwickelte und anpassbare HTML5, CSS und JavaScript-Framework „Bootstrap“ als Meteor-Paket eingesetzt. Dadurch ist

es möglich, mit unterschiedlichen, vordefinierten Designelementen und JavaScript-Funktionen eine einheitliche, dynamische und benutzerfreundliche Web - Oberfläche zu erstellen, die den Anforderungen an die Benutzerfreundlichkeit genügt. Damit ist die Web - Oberfläche übersichtlich und strukturiert gehalten.

7.5.2 Schutz vor unbefugter Ausführung und Änderung

Die Aktualisierung von externen Marktdaten kann nur durch die Übergabe als Parameter von einer zuvor vertraulichen Partei erfolgen. Die für den Geschäftsvorfall benötigten externen Marktdaten können nur von einer zuvor definierten Oracle-Adresse im Ethereum-Netzwerk aktualisiert werden. Die Realisierung erfolgt durch eine Prüfung der Senderadresse der jeweiligen Transaktion für den Aufruf der Funktion des Smart Contracts. Diese Adresse wird zuvor intern in der Instanz des Smart Contracts gespeichert. Diese Vorgehensweise entspricht dem „Restricting Access“ - Pattern von „Solidity“ (EthereumG, 2015).

7.5.3 Aufsetzen des Derivat-Vertrags

Durch die Zweiteilung der DAPP, in die Erstellung und dem Abruf sowie der Interaktion mit einer Smart Contract-Instanz ist es möglich, dass Nutzer der DAPP weitere Derivat-Verträge mit definierten Parametern aufsetzen können. Dazu werden in einer Eingabemaske die Parameter erstellt und nach einem Klick auf den Button eine Instanz des Smart Contracts im Ethereum-Netzwerk erzeugt. Dabei ist zu beachten, dass der Kontrakt erst nach dem Mining durch die Nutzer von „Ethereum“ erstellt wird. Um eine größtmögliche Benutzerfreundlichkeit in der DAPP zu bieten, wird die Adresse eines zuvor erstellten Smart Contracts in einer leichtgewichtigen MongoDB[17], die im Meteor-Framework integriert ist, gespeichert. Zum einen muss der Benutzer dadurch nicht selber die Adresse abspeichern, zum anderen wird ihm die Möglichkeit gegeben, in der Informationssicht komfortabel den jeweiligen Smart Contract auszuwählen.

Des Weiteren ist nach dem Deployment zu beachten, dass die Instanz zwar anschließend in Ethereum erstellt, aber der eigentliche Vertrag nicht aktiviert worden ist. Der Smart Contract wird erst aktiviert, wenn beide Vertragsparteien ihre jeweilige initiale Margin eingezahlt haben. Dies erfolgt über eine separate Funktion (siehe Kapitel 7.5.4). Hinzukommt, dass die Transaktionskosten für das Deployment vom Ersteller übernommen werden müssen.

[17] MongoDB ist eine dokumentenorientierte, NO-SQL Datenbank, welche in Meteor integriert ist, https://www.mongodb.com/de.

Eine weitere Einschränkung ist, dass der Ersteller des Smart Contracts automatisch der Käufer in Bezug auf den Derivat-Vertrag darstellt. Durch eine generischere Implementierung ist es möglich, dass die Gegenpartei als Käufer definiert wird. Für den Prototyp ist aufgrund des durch diese Restriktion vereinfachten Testens, dieser Fall vorerst nicht betrachtet worden.

Nach dem Deployment des Smart Contracts ist es ebenso notwendig, dass eine der Vertragsparteien die Registrierung am Oracle ausführt, damit die Aktualisierung des entsprechenden Marktpreises im Smart Contract erfolgen kann. Dies ist nicht im Deployment vorgesehen, sondern in eine separate Funktion ausgegliedert worden, weil eine der Vertragsparteien beim Aufruf eine bestimmte Summe an „Ether“ mitsenden muss. Diese Summe dient der Bezahlung für die Aktualisierung des Preises, da dies nur über eine Transaktion in Ethereum realisiert werden kann. Die hat zur Folge, dass das Oracle diese Transaktionskosten übernehmen muss. Die Implementierung der Registrierungsfunktion ist in Kapitel 7.5.8 beschrieben.

7.5.4 Einzahlung von Margin

Die auszuführenden Operationen für die initiale Margin zur Aktivierung des Vertrags, als auch der Margin Call unterscheiden sich nur dadurch, dass nach der Einzahlung der initialen Margin gleichzeitig die Laufzeit des Vertrages beginnt. Anschließend kann die fortlaufende Aktualisierung der externen Marktdaten, als auch der Kontostände der beiden Vertragsparteien erfolgen. Ansonsten wird in beiden Fällen geprüft, welche der beiden Vertragsparteien die Margin und in welcher Höhe überwiesen hat. Sollte die Höhe der Einzahlung dem definierten Wert entsprechen, so wird der jeweilige interne Kontostand angepasst. Dabei richtet sich die Höhe an zuvor fest definierten Werten. Im Fall der initialen Margin ist dies üblicherweise 10 Prozent des Vertragsvolumens. Beim Margin Call ist die Höhe von der Differenz des aktuellen internen Kontostands zur initialen Margin abhängig. Ansonsten wird die überwiesene Summe an den Sender zurücküberwiesen. Eine zusätzliche Besonderheit beim Margin Call ist, dass das Flag „MarginCallRequested" bei der jeweiligen Partei wieder auf „false“ gesetzt wird.

7.5.5 Auflösen des Derivat-Vertrags

Das Settlement erfolgt, wie bereits in Kapitel 5.2 erwähnt, nach Ablauf der zu Beginn definierten Laufzeit oder wenn eine der beiden Vertragsparteien seiner Pflicht zum Nachschuss, dem Margin Call, nach einer bestimmten Frist nicht nachkommen sollte. Nach Ablauf einer dieser beiden Fristen ist es möglich, dass von einer der beiden Vertragsparteien die Settlement-Funktion der Instanz aufgerufen werden kann. Dies wird über zwei Modifier im Smart Contract gewährleistet. Ist dies der Fall, so werden die internen Konten aufgelöst

und die Höhe der Kontostände auf die jeweiligen Konten der Vertragsparteien in „Ethereum“ überwiesen. Zu beachten ist hierbei, dass beim Settlement nach einem nicht erfolgten Margin Call, die Höhe der Auszahlung davon abhängt, ob die Vertragspartei, die den Margin Call durchführen muss, eine negative Bilanz hat. Sollte dies der Fall sein, so wird der Gegenpartei die gesamte Menge an „Ether“, die im Smart Contract vorhanden ist, ausgezahlt. Ist dies nicht der Fall, so wird jeder Vertragspartei der jeweilige, aktuelle Kontostand ausgezahlt. Dies wird durch eine Historisierung des vorherigen Kontostandes im Participant-Objekt ermöglicht. Abschließend wird die Instanz durch die interne Funktion „suicide“[18] im Ethereum-Netzwerk abgebaut, sodass diese nicht mehr verfügbar bzw. abrufbar ist. Die sich noch in der Instanz befindliche „Ether“ werden an einen Account gesendet werden. Als Parameter für die suicide-Funktion wird die Adresse des Erstellers des Smart Contracts in Ethereum übergeben, da dieser für die Kosten des Deployments des Smart Contracts in Vorleistung gehen muss.

7.5.6 Abruf von internen Vertragsdaten

Jegliche Vertragsdaten, als auch externe, für den Vertrag relevante Marktdaten, werden intern im Smart Contract gespeichert. Dazu zählen unter anderem Vertragsdaten wie z.B. Vertragsvolumen, Margin, Daten der Vertragsparteien wie z.B. der Kontostand als auch externe Marktdaten wie z.B. der Preis zu einem bestimmten Zeitpunkt von einer bestimmten Börse. Die Daten sind übersichtlich und in drei Kategorien über die DAPP zu jedem Zeitpunkt abrufbar. Die Daten liegen über Transaktionen verteilt auf der unterliegenden Blockchain im Ethereum-Netzwerk und sind der Instanz des erstellten Smart Contract zugeordnet. Somit liegen die Daten als Kopie auch bei den Vertragsparteien vor, sodass keine Transaktionskosten anfallen und kein Transaktion mit abschließendem „Mining“ für den Abruf nötig ist. Des Weiteren handelt es sich dabei um einfache Getter-Methoden innerhalb des Smart Contracts, da die entsprechenden Variablen als public definiert sind. Die Deklaration der Variablen als private ist möglich, schränkt aber nicht die Sichtbarkeit und den Zugriff auf diese ein, da sämtliche Daten bzw. Inhalte von Transaktionen für jeden Teilnehmer im Ethereum-Netzwerk öffentlich sichtbar sind. Dieser Umstand ist durch die darunterliegende Technologie und dem Prinzip einer Blockchain gegeben.

[18] Die suicide-Funktion ist ein von Ethereum bzw. Solidity bereitgestellte Funktion zum Abbau einer Instanz eines Smart Contracts. Sie dient außerdem dem Überweisen von noch im Vertrag befindlichen „Ether“.

7.5.7 Aktualisierung von Marktdaten durch Dritte

Für den Erhalt von externen Marktdaten ist eine Instanz des Clearinghouse-Smart Contracts auf eine oder mehrere Oracle-Smart Contracts in „Ethereum“ angewiesen. Die Aktualisierung der internen Marktdaten im Smart Contract kann dabei über zwei unterschiedliche Abläufe erfolgen. In beiden Fällen muss die Instanz von außen reaktiviert werden, damit die entsprechenden Operationen ausgeführt werden können. Zum einen kann eine der Vertragsparteien die Aktualisierungsfunktion über die DAPP aufrufen, wodurch anschließend die Instanz, die Funktion zum Abruf eines bestimmten Datums eines definierten Oracles aufruft und so den gewünschten Wert erhält. Der Vorteil ist dabei, dass die Vertragsparteien nur beim jeweiligen Aufruf für die Transaktionskosten aufkommen müssen.

Andererseits kann es unter Umständen in diesem Szenario dazu kommen, dass z.B. der Preis an einem Tag nicht aktualisiert wird. Aus diesem Grund wird das zweite Szenario, welches durch Abbildung 33 veranschaulicht wird, für die Aktualisierung von externen Marktdaten angewendet. Dabei wird, wie bereits erwähnt, die jeweilige Instanz an einem Oracle registriert. Das Oracle ruft in einem bestimmten Intervall die Funktion zur Aktualisierung des Preises auf. So ist gewährleistet, dass z.B. der Preis und die Kontostände unabhängig von den beiden Vertragsparteien aktualisiert werden. Die Transaktionskosten für den mehrmaligen Aufruf der Funktion zur Aktualisierung im Smart Contract werden zu Beginn der Laufzeit von einer der Vertragsparteien der Instanz überwiesen und anschließend bei der Registrierung dem Oracle weitergeleitet.

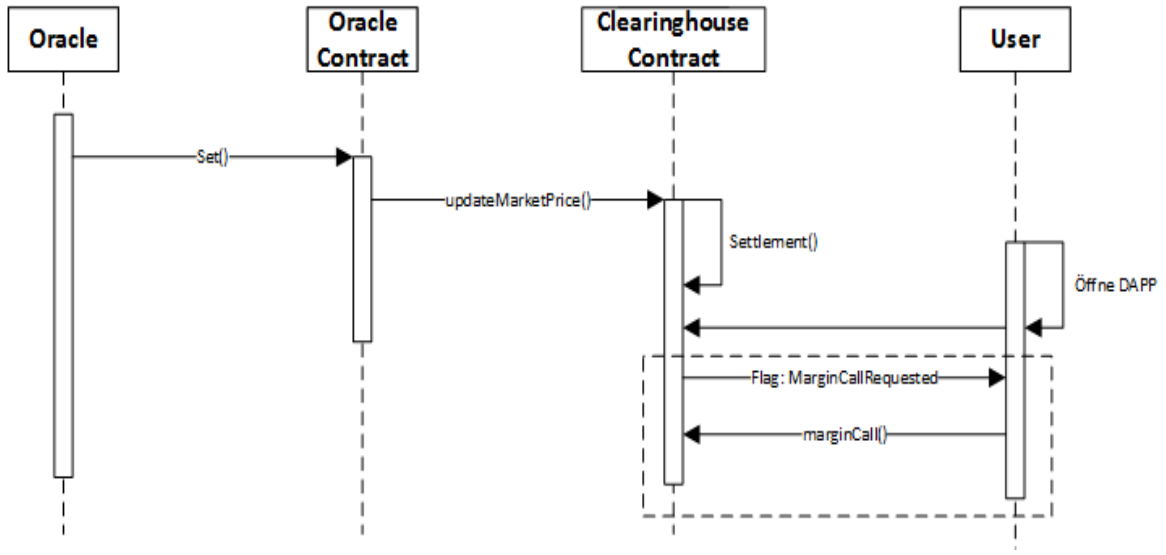

Abbildung 33: Oracle initiiert Preisaktualisierung

Des Weiteren erfolgt nach der Aktualisierung des Preises die Berechnung der internen Kontostände der Vertragsparteien, das Clearing (siehe Kapitel 5.2). Sollte der berechnete interne Kontostand nach dem Clearing unterhalb einer zuvor definierten „Maintenance

Margin“ fallen, so wird das Flag „MarginCallRequested“ gesetzt Dem jeweiligen Benutzer wird dann in der DAPP angezeigt, dass ein Nachschuss erfolgen muss. Gleichzeitig wird ebenfalls die Ablauffrist „deadlineMarginCall“ auf einen definierten Zeitpunkt gesetzt bis zu dem der Margin Call erfolgen muss. Außerdem wird das Flag „MarginCallFreezed“ gesetzt, sodass in diesem Zeitraum keine Aktualisierung von externen Marktdaten erfolgen kann.

7.5.8 Registrierung am Oracle

Die Registrierung der Smart Contract-Instanz an einem Oracle besteht nur aus einer Transaktion die von der Instanz an das Oracle gesendet wird. Diese Funktionalität ist in einer separaten Funktion und nicht bei dem Deployment des Smart Contracts realisiert. Sie muss explizit durch eine der Vertragsparteien vor der Einzahlung der initialen Margin aufgerufen werden. Dies ist damit begründet, dass bei der Registrierung eine bestimmte Summe an „Ether“ an das Oracle überwiesen werden muss, welche als Bezahlung der Transaktionen für die Aktualisierung des Preises dient. Dieser Umstand hat ebenfalls zur Folge, dass eine der Vertragsparteien in Vorauszahlung geht und damit mehr „Ether“ aufwenden muss als die Gegenpartei. Die Transaktion beinhaltet die Adresse der Instanz des Smart Contracts sowie die Summe an „Ether“, die zur Bezahlung der Transaktionskosten nötig ist. Wie in Kapitel 7.3.3 ausgeführt, erfolgt eine Aktualisierung nur solange wie genügend „Ether“ zur Verfügung gestellt wird. Dadurch kann es unter Umständen dazu kommen, dass keine Aktualisierung des Preises nach einer bestimmten Laufzeit erfolgt. Aus dem diesem Grund hängt die Summe an „Ether“, die mit gesendet wird, von der Laufzeit des Vertrages ab. Die Höhe muss von den Vertragsparteien in Abhängigkeit der Laufzeit und der Kosten für diese Transaktion selbst errechnet und als Wert bei der Registrierung eingetragen werden. Eine Nachzahlung für weitere Preisaktualisierung ist im gegenwärtigen Kontrakt nicht implementiert, sodass genügend „Ether“ zu Beginn überwiesen werden muss.

7.6 Kritische Reflexion und Ausblick

Der größte Kritikpunkt liegt in der Verarbeitung von Zahlen. Im Smart Contract bzw. der DAPP sind keine Gleitkommazahlen vorgesehen, sondern die Preise müssen exakt auf eine kleinere Einheit umgerechnet worden sein. Intern findet dieser Umrechnungsschritt im Smart Contract oder in der DAPP nicht statt. Die Einschränkungen bezüglich der Verwendung von ausschließlich ganzen Zahlen hat ihren Ursprung in der Benutzung von Solidity als Programmiersprache. Diese sieht keine Gleitkommazahlen-Datentypen vor. In Ethereum werden jegliche Währungen z.B. von „Ether“ in die kleinste Einheit „Wei“ umgerechnet. Dies hat zur Folge, dass mit sehr großen Zahlen gerechnet werden muss.

Ein inhaltlicher Kritikpunkt ist, dass die Werte für die initiale Margin mit zehn Prozent, als auch die „Maintenance Margin“ mit fünf Prozent des zugrundeliegenden Vertragsvolumens bei der

Erstellung des Smart Contracts im Quellcode fest definiert sind. Damit soll gewährleistet werden, dass keine Gleitkommazahlen wegen der zuvor geschilderten Einschränkung bezüglich „Solidity“ vorhanden sind. Durch eine Parametrisierung können diese Werte dynamisch gehalten werden. Dabei bleibt allerdings das Problem mit den Gleitkommazahlen bestehen und führt unter Umständen zu ungewollten Rundungen.

Ein weiterer Kritikpunkt sind die teilweise umständlichen Workflows. Hier ist das größte Verbesserungspotential vorhanden. Zum Beispiel sollte die Registrierung am Oracle nicht separat durch eine der Vertragsparteien aufgerufen werden müssen, sondern schon zum Zeitpunkt des Deployments des Smart Contracts erfolgen. Des Weiteren kann es unter Umständen dazu kommen, dass ein Vertrag zu spät aufgelöst wird, wenn z.B. eine der Vertragsparteien seinem Margin Call nicht nachkommt, da die dafür zuständige Funktion explizit durch einen der Vertragsparteien im Smart Contract aufgerufen werden muss. Allerdings ist dieser Umstand durch die Passivität von Smart Contract-Instanzen bedingt, wodurch es vorerst keine Lösung gibt.

Als Erweiterung kann eine Historisierung von Marktpreisen sowie der einzelnen Kontostände in der Instanz des Smart Contracts in Betracht gezogen werden. Dadurch ist es möglich, dass durch entsprechende Visualisierung mit einer Grafik-Bibliothek wie z.B. C3.js[19], die Entwicklung im Zeitverlauf übersichtlich und komfortabel den Vertragsparteien präsentiert werden kann.

[19] C3.js, eine auf D3.js basierende Grafik-Bibliothek, http://c3js.org.

8 Fazit

Die Blockchain als Basis einer Kryptowährung oder auch als Basis integritätsgesicherter r Daten innerhalb von erweiterten Transaktionen kann je nach Einsatzgebiet, verwendeter Sicherheitsrichtlinien und Art der Implementierung nahezu absolute Sicherheit, oder auch einfachste Angriffsmöglichkeiten bieten.

Seit 2009 entwickelt sich die Blockchain-Technologie stetig weiter. Wie bei jeder neuen Technologie tauchen mit zunehmender Verbreitung neue Problematiken auf. Die erforderliche Skalierbarkeit wird dabei für bereits jetzt große Kryptowährungen wie den Bitcoin eine Herausforderung. Hierunter fallen Limitierungen in der maximalen Aufnahme von Transaktionen pro Sekunde, sowie die steigende Anforderung an Rechenkapazität für die Sicherheit des Netzwerks.

Der Einsatz neuester Kryptografieverfahren, wie dem SHA3 Hash im Rahmen des Proof-of-Work, kann sich wie im Bitcoin-Netzwerk positiv auf die Sicherheit auswirken und gleichsam einen äußerst verschwenderischen Umgang mit Ressourcen mit sich bringen. Der Einsatz alternativer Implementierungen, beispielsweise mit Proof-of-Stake, vermindert den Ressourceneinsatz unter Schaffung von Vertrauensverhältnissen gegenüber vermögenden Teilnehmern. Die stetige Entwicklung an weiteren teils hybriden Konzepten wirkt sich dabei positiv auf die Sicherheit und auch den Energieverbrauch aus.

Trotz berechtigter Kritik bietet die Blockchain-Technologie, nicht nur für den Einsatz in Bezug auf digitale Währungen, großes Potential. So können ebenfalls auf der Blockchain-Technologie reale Geschäftsvorfälle durch Smart Contracts realisiert werden. Die Smart Contract-Plattformen und Programme für Smart Contracts befinden sich in einem frühen Stadium der Entwicklung. Es gibt viele Ideen und Projekte, allerdings sind diese zum gegenwärtigen Zeitpunkt bei weitem noch nicht für Endbenutzer geeignet. Einige Finanzdienstleister nehmen sich diesem Thema vermehrt an. Auch mit der Verbreitung des Begriffs „Internet of Things“ gibt es ebenfalls viele Anwendungsfälle, die außerhalb der Finanzbranche möglich sind. Allerdings stehen diese Forschungen und Entwicklungen großen Herausforderungen wie z.B. der rechtlichen und physischen Durchsetzbarkeit gegenüber. Einen Nutzen haben Smart Contracts überwiegend dort, wo schädliche Manipulationen möglich sind oder einem Mittelsmann vertraut werden muss. In solchen Anwendungsfällen können Smart Contracts aufgrund ihrer Dezentralität und dem im Quellcode fest definierten Verhalten, das Risiko minimieren.

Komplexe, wie in der Ausarbeitung entwickelte und implementiere, Geschäftsvorfälle sind nur mit Einschränkungen realisierbar. So ist z.B. der Bezug von externen Daten außerhalb

der Blockchain nur über Umwege, wie ein Smart Oracle, möglich. Einfache Geschäftsvorfälle, wie z.B. ein Treuhand-Vertrag, sind dahingegen uneingeschränkt realisierbar.

Die Auswahl des Geschäftsvorfalls hat eine hohe Praxisrelevanz. Dies zeigt sich durch die beiden US-amerikanischen Startup-Unternehmen Hitfin und Hedgy, welche parallel zu diesem Projekt Lösungen für den Derivatehandel auf Basis der Blockchain entwickeln.

Literaturverzeichnis

Ateniese, G., Kamara, S., & Katz, J. (2009). Proofs of Storage from Homomorphic Identification Protocols. Abgerufen am 10. Dezember 2015 von https://cs.umd.edu/~jkatz/papers/pdp.pdf

Back, A. (01. August 2002). Hashcash - A Denail of Service Counter-Measure. *Hashcash - A Denail of Service Counter-Measure*. Von Hashcash.org: http://hashcash.org/ abgerufen

Bentov, I., Lee, C., Mizrahi, A., & Rosenfeld, M. (2014). Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake. Abgerufen am 05. November 2015 von https://eprint.iacr.org/2014/452.pdf

Bertani, T. (4. November 2015). *Oraclize Blog.* Abgerufen am 11. November 2015 von http://blog.oraclize.it/2015/11/04/oraclize-official-launch/

Berwanger, J. (kein Datum). *Gabler Wirtschaftslexikon - juristische Person*. Abgerufen am 25. Januar 2016 von http://wirtschaftslexikon.gabler.de/Definition/juristische-person.html

Bitcoin.org. (2015a). *Double-spending*. Abgerufen am 10. Februar 2016 von bitcoinwiki: https://en.bitcoin.it/wiki/Double-spending

Bitcoin.org. (2015b). *Weaknesses*. Abgerufen am 05. Januar 2016 von bitcoinwiki: https://en.bitcoin.it/wiki/Weaknesses

Bitcoin.org-A. (kein Datum). *Bitcoin.org - Developer Guide - Blockchain*. Abgerufen am 10. 02 2016 von Bitcoin.org - Developer Guide - Blockchain: https://bitcoin.org/en/developer-guide#block-chain

Bitcoin.org-B. (kein Datum). *Bitcoin.org - Wie es funktioniert*. Abgerufen am 10. 02 2016 von Bitcoin.org - Wie es funktioniert: https://bitcoin.org/de/wie-es-funktioniert

Bitcoinwiki-A. (kein Datum). *Bitcoinwiki - Bitcoin Core*. Abgerufen am 10. 02 2016 von Bitcoinwiki - Bitcoin Core: https://en.bitcoin.it/wiki/Bitcoin_Core

Bitcoinwiki-B. (kein Datum). *Bitcoinwiki - Block hashing algorithm*. Abgerufen am 10. 02 2016 von Bitcoinwiki - Block hashing algorithm: https://en.bitcoin.it/wiki/Block_hashing_algorithm

Bitcoinwiki-C. (kein Datum). *Bitcoinwiki - Blockchain*. Abgerufen am 10. 02 2016 von Bitcoinwiki - Blockchain: https://en.bitcoin.it/wiki/Block_chain

Bitcoinwiki-D. (10. Februar 2015). *Bitcoinwiki - ECDSA*. Abgerufen am 10. Februar 2016 von Bitcoinwiki - ECDSA: https://en.bitcoin.it/wiki/Elliptic_Curve_Digital_Signature_Algorithm

Bitcoinwiki-E. (kein Datum). *Bitcoinwiki - Hashcash*. Abgerufen am 10. 02 2016 von Bitcoinwiki - Hashcash: https://en.bitcoin.it/wiki/Hashcash

Bitcoinwiki-F. (kein Datum). *Bitcoinwiki - Message Structure*. Abgerufen am 10. 02 2016 von Bitcoinwiki - Message Structure: Bitcoinwiki - Message Structure

Bitcoinwiki-G. (kein Datum). *Bitcoinwiki - Mining*. Abgerufen am 10. 02 2016 von Bitcoinwiki - Mining: https://en.bitcoin.it/wiki/Mining

Bitcoinwiki-H. (kein Datum). *Bitcoinwiki - Proof-of-Work*. Abgerufen am 10. 02 2016 von Bitcoinwiki - Proof-of-Work: https://bitcoin.org/en/developer-guide#proof-of-work

Bitcoinwiki-I. (kein Datum). *Bitcoinwiki - Script Flow Control*. Von Bitcoinwiki - Script Flow Control: https://en.bitcoin.it/wiki/Script#Flow_control abgerufen

Bitcoinwiki-K. (kein Datum). *Bitcoinwiki - Target*. Abgerufen am 10. Februar 2016 von Bitcoinwiki - Target: https://en.bitcoin.it/wiki/Target

Bitcoinwiki-L. (28. Mai 2015). *Bitcoinwiki - Transaktionen*. Abgerufen am 10. Februar 2016 von Bitcoinwiki - Transaktionen: https://en.bitcoin.it/wiki/Transaction

Bitcoinwiki-M. (2016). *Bitcoin Wiki - Script*. Abgerufen am 10. 02 2016 von https://en.bitcoin.it/wiki/Script

Bitcoinwiki-N. (2016). *Bitcoin Wiki - Block size limit controversy*. Abgerufen am 26. Februar 2016 von https://en.bitcoin.it/wiki/Block_size_limit_controversy

Blockchaincenter. (2016). *Blockchaincenter - Blockchaingröße*. Abgerufen am 10. Februar 2016 von Blockchaincenter - Blockchaingröße: http://www.blockchaincenter.de/fragen/wie-gross-ist-die-blockchain/

Boehm, F., & Pesch, P. (2014). Bitcoin: A First Legal Analysis. *1st Workshop on Bitcoin Research in Assocation with Financial Crypto*.

Burst. (2015). *Burst*. Abgerufen am 01. Februar 2016 von http://burstcoin.info/

CoingeckoA. (kein Datum). *Coingecko.com.* Abgerufen am 26. Januar 2016 von https://www.coingecko.com/en/coins/bitcoin

CoingeckoB. (kein Datum). *Conigecko.com.* Abgerufen am 26. Januar 2016 von https://www.coingecko.com/en/coins/ethereum

Coinmarketcap. (kein Datum). *Coinmarketcap.com*. Abgerufen am 30. September 2015 von Coinmarketcap.com: http://coinmarketcap.com/all/views/all/

Cordell, D. (30. Oktober 2014). HDD Poof of Capacity Mining, is it viable? Abgerufen am 17. Januar 2016 von https://www.cryptocoinsnews.com/hdd-proof-of-capacity-mining-viable/

Counism. (23. März 2014). Why Proof-of-Burn. Abgerufen am 05. Januar 2016 von http://counterparty.io/news/why-proof-of-burn/

Decker, C., & Wattenhofer, R. (2013). Information Propagation in the Bitcoin Network. *Information Propagation in the Bitcoin Network*.

Decker, C., & Wattenhofer, R. (2013). Information Propagation in the Bitcoin Network. Zurich, Schweiz. Abgerufen am 25. Oktober 2015 von http://diyhpl.us/~bryan/papers2/bitcoin/Information%20propagation%20in%20the%20Bitcoin%20network.pdf

Eikenberg, R. (20. Dezember 2015). *l+f: Bitcoin-App Blockchain generierte Gemeinschaftskonto.* (h. Security, Hrsg.) Abgerufen am 30. Dezember 2015 von http://www.heise.de/security/meldung/l-f-Bitcoin-App-Blockchain-generierte-Gemeinschaftskonto-2678065.html

Eris Industries. (2016). *Explainer - Permissioned Blockchains*. Abgerufen am 26. Februar 2016 von https://docs.erisindustries.com/explainers/permissioned_blockchains

Ernst, N. (16. Juni 2014). *51 Prozent der Rechenleistung für Bitcoin in einer Hand.* (golem.de, Hrsg.) Abgerufen am 03. Januar 2016 von http://www.golem.de/news/ghash-51-prozent-der-rechenleistung-fuer-bitcoin-in-einer-hand-1406-107217.html

Ethereum. (21. November 2015). *Ethereum Whitepaper.* Abgerufen am 18. Januar 2016 von https://github.com/ethereum/wiki/wiki/White-Paper#messages-and-transactions

EthereumC. (2015). *Ethereum Whitepaper - A Next-Generation Smart Contract and Decentralized Application Platform*. Abgerufen am 03. Februar 2016 von https://github.com/ethereum/wiki/wiki/White-Paper

EthereumD. (2015). *Ethereum Whitepaper - Code Execution*. Abgerufen am 03. Februar 2016 von https://github.com/ethereum/wiki/wiki/White-Paper#code-execution

EthereumD. (2015). *Go-Ethereum Wiki*. Abgerufen am 03. Februar 2016 von https://github.com/ethereum/go-ethereum/wiki

EthereumE. (2015). *Mist Browser*. Abgerufen am 03. Februar 2016 von https://github.com/ethereum/mist

EthereumF. (2015). *Mix: The DApp IDE*. Abgerufen am 03. Februar 2016 von https://github.com/ethereum/wiki/wiki/Mix:-The-DApp-IDE

EthereumG. (2015). *Solidity - Restricting Access*. Abgerufen am 09. Februar 2016 von http://solidity.readthedocs.org/en/latest/common-patterns.html#restricting-access

EthereumH. (2015). *Solidity Documentation*. Abgerufen am 03. Februar 2016 von https://solidity.readthedocs.org/en/latest/

Europäische Zentralbank. (kein Datum). *Euro foreign exchange reference rates.* Abgerufen am 20. Januar 2016 von https://www.ecb.europa.eu/stats/exchange/eurofxref/html/index.en.html

Eurostat. (22. Januar 2016). *Electricity production, consumption and market overview.* Abgerufen am 10. Februar 2016 von http://ec.europa.eu/eurostat/statistics-explained/index.php/Electricity_production,_consumption_and_market_overview

Gray, M. (2015). Abgerufen am 26. Januar 2016 von Ethereum Blockchain as a Service now on Azure: https://azure.microsoft.com/de-de/blog/ethereum-blockchain-as-a-service-now-on-azure/

HeldtA, C. (kein Datum). *Gabler Wirtschaftslexikon - Differenzgeschäft*. Abgerufen am 18. Januar 2016 von http://wirtschaftslexikon.gabler.de/Archiv/4512/differenzgeschaeft-v9.html

HeldtB, C. (kein Datum). *Gabler Wirtschaftslexikon - Kursrisiko.* Abgerufen am 11. Januar 2016 von http://wirtschaftslexikon.gabler.de/Definition/kursrisiko.html

Honsel, G. (2015). *Bitcoin: Die Banken drehen den Spieß um*. Abgerufen am 26. Januar 2016 von http://www.heise.de/newsticker/meldung/Bitcoin-Die-Banken-drehen-den-Spiess-um- 2825544.html

Hull, J. (2015). *Optionen, Futures und andere Derivate.* Hallbergmoos: Pearson.

ITWissen. (11. Februar 2016). *Zugriffszeit.* Abgerufen am 11. Februar 2016 von ITWissen Das große Online-Lexikon für Informationstechnologie: http://www.itwissen.info/definition/lexikon/Zugriffszeit-access-time.html

Johnston, D., Onat Yilmaz, S., Kandah, J., Bentenitis, N., Hashemi, F., Gross, R., . . . Mason, S. (2014). *The General Theory of Decentralized Applications, Dapps*. Abgerufen am 20. Januar 2016 von https://github.com/DavidJohnstonCEO/DecentralizedApplications

Kannenberg, A. (16. Februar 2015). *Angriffsziel Bitcoinbörse: Bter und Exco.in gehackt.* (h. online, Hrsg.) Abgerufen am 28. Dezember 2015 von http://www.heise.de/newsticker/meldung/Angriffsziel-Bitcoinboerse-Bter-und-Exco-in-gehackt-2550175.html

Kosba, A., Miller, A., Shi, E., Wen, Z., & Papamanthou, C. (2015). *Hawk: The blockchain model of cryptography and privacy-preserving smart contracts.* University of Maryland and Cornell University.

Koschyk, H. (27. September 2013). *www.frank-schaeffler.de.* Von http://www.frank-schaeffler.de/wp-content/uploads/2013/10/2013_09_27-Antwort-Koschyk-Bitcoin3.pdf abgerufen

Maxim, J. (2015). *Ripple Discontinues Smart Contract Platform Codius, Citing Small Market*. Abgerufen am 20. Januar 2016 von https://bitcoinmagazine.com/articles/ripple-discontinues-smart-contract-platform-codius-citing-small-market-1435182153

Miller, S. M., & Stiegler, M. (2003). *The Digital Path: Smart Contracts and the Third World*. Abgerufen am 26. Januar 2016 von http://www.erights.org/talks/pisa/paper

Münzer, J. (19. Dezember 2013). *BaFin*. Von http://www.bafin.de/SharedDocs/Veroeffentlichungen/DE/Fachartikel/2014/fa_bj_1401_bitcoins.html abgerufen

Nakamoto, S. (2009). Bitcoin: A Peer-to-Peer Electronic Cash System. *Bitcoin: A Peer-to-Peer Electronic Cash System*.

Nxt Community. (2015). *Nct*. Abgerufen am 02. Februar 2016 von http://nxt.org/

O'Dwyer, K. J., & Malone, D. (2014). Bitcoin Mining and its Energy Footprint. (H. I. Maynooth, Hrsg.) Abgerufen am 28. Dezember 2015 von https://karlodwyer.github.io/publications/pdf/bitcoin_KJOD_2014.pdf

O'Dwyer, K. J., & Malone, D. (26-27. Juni 2014). Bitcoin Mining and its Energy Footprint. *Bitcoin Mining and its Energy Footprint*. Limerick.

Oracle. (2010). *Class String.* Abgerufen am 08. Januar 2016 von docs.oracle.com: http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html#hashCode%28%29

Orisi. (29. November 2014). *Orisi White Paper.* Von https://github.com/orisi/wiki/wiki/Orisi-White-Paper abgerufen

Orisi. (17. Januar 2016). *The Oracle List.* Von http://oracles.li/# abgerufen

P4Titan. (17. Mai 2014). Slimcoin A Peer-to-Peer Crypto-Currency with Proof-of-Burn. Abgerufen am 10. Februar 2016 von https://decentralgroup.com/archive_files/whitepaper%20(4).pdf

Patterson, R. (26. August 2015). *Alternatives for Proof of Work, Part 2: Proof of Activity, Proof of Burn, Proof of Capacity, and Byzantine Generals.* Abgerufen am 30. Dezember 2015 von https://bytecoin.org/blog/proof-of-activity-proof-of-burn-proof-of-capacity/

Peercoin. (14. März 2015). *Peercoin wiki*. Abgerufen am 08. Februar 2016 von https://wiki.peercointalk.org/index.php?title=Main_Page

QuantumMechanic. (11. Juli 2011). *Proof of stake instead of proof of work.* Abgerufen am 14. Oktober 2015 von Bitcoin Forum: https://bitcointalk.org/index.php?topic=27787.0

Reality KeysA. (kein Datum). *Reality Keys - About.* Abgerufen am 19. Januar 2016 von https://www.realitykeys.com/about

Reality KeysB. (kein Datum). *Reality Keys - Developer Resources and Examples.* Abgerufen am 19. Januar 2016 von https://www.realitykeys.com/developers/resources#ethereum

Reality KeysC. (kein Datum). *Reality Keys - Pricing.* Abgerufen am 16. Januar 2016 von https://www.realitykeys.com/pricing

Ripple Labs. (17. Juli 2014). *Smart Oracles: A Simple, Powerful Approach to Smart Contracts.* Abgerufen am 19. Januar 2016 von https://github.com/codius/codius/wiki/Smart-Oracles:-A-Simple,-Powerful-Approach-to-Smart-Contracts

Schmeh, K. (2013). *Kryptograpfie - Verfahren, Protokolle, Infrastrukturen.* Heidelberg: dpunkt Verlag.

Schöning, S. (kein Datum). *Gabler Wirtschaftslexikon - Währungsrisiko*. Abgerufen am 18. Januar 2016 von http://wirtschaftslexikon.gabler.de/Archiv/9441/waehrungsrisiko-v9.html

Swan, M. (2015). *Blockchain - Blueprint for a New Economy.* O'Reilly Media Verlag.

Szabo, N. (1998). *Formalizing and Securing Relationships on Public Networks*. Abgerufen am 26. Januar 2016 von http://szabo.best.vwh.net/formalize.html

TLSNotary. (11. November 2015). *TLSNotary.* Von https://tlsnotary.org/ abgerufen

von Uhruh, C.-N. (2015). *The Law of Bitcoin.* Bloomington: iUniverse.

Vossen, G., & Weikum, G. (2001). *Fundamentals of Transactional Information Systems.* San Diego: Academic Press.

Wood Dr., G. (2014). *Ethereum - A secure decentralised generalised Transaction Ledger*. Abgerufen am 09. Februar 2016 von http://gavwood.com/paper.pdf

Anhang

A. Analyse aktueller Kryptowährungen

Die Ergebnisse der Analyse der Kryptowährungen liegen als Excel-Datei „Anhang A - Analyse aktueller Kryptowährungen.xlsx“ vor.

B. Transaktionen

Die Beispiele und Visualisierungen zum Bereich Transaktionen liegen als Visio-Datei „Anhang B-E Transaktionen, Blöcke, Dezentrales Netzwerk, Kryptografie.vsdx“ vor.

B.1 – Transaktionen im Zeitverlauf (Verknüpfung von Input und Output)

B.2 – Prüfung einer Transaktionssignatur

B.3 – Erstellen eines Transaktionshashes

C. Blöcke

Die Beispiele und Visualisierungen zum Bereich Blöcke liegen als Visio-Datei „Anhang B-E Transaktionen, Blöcke, Dezentrales Netzwerk, Kryptografie.vsdx“ vor.

C.1 – Berechnung des Hashbaums (Merkle Root)

C.2 – Berechnung des Schwierigkeitsgrades

C.3 – Hashcash-Algorithmus

C.4 – Prüfen eines Blockhashes

C.5 – Blockgenerierung im Zeitverlauf

D. Dezentrales Netzwerk

Die Beispiele und Visualisierungen zum Bereich Dezentrales Netzwerk liegen als Visio-Datei „Anhang B-E Transaktionen, Blöcke, Dezentrales Netzwerk, Kryptografie.vsdx“ vor.

D.1 – Verbinden mit dem Netzwerk

D.2 – Initialer Blockdownload

D.3 – Transaktionen senden und empfangen

E. Kryptografie

Die Beispiele und Visualisierungen zum Bereich Kryptografie liegen als Visio-Datei „Anhang B-E Transaktionen, Blöcke, Dezentrales Netzwerk, Kryptografie.vsdx“ vor.

E.1 – Digitale Signaturen (Beispiel RSA)

E.2 – Kryptografische Hashfunktion (SHA256)

F. Entwicklungsumgebung für den Blockchain-Prototyp

Typ	Name	Version	Quelle/Link
Programmiersprache	Java	1.8.0_66	http://www.oracle.com/de/java/overview/index.html
Java Bibliothek	java.net	Java 1.7	https://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html
Java Bibliothek	java.nio.file	Java 1.7	https://docs.oracle.com/javase/7/docs/api/java/nio/file/package-summary.html
Java Bibliothek	org.json	20141113	http://www.json.org/
Framework	Log4j	1.2.17	http://logging.apache.org/log4j/
Entwicklungsumgebung	Eclipse Luna	4.4.2	Eclipse https://eclipse.org/luna/
Build-Management-Tool	Apache Maven	3.2.1	https://maven.apache.org/

Tabelle 5: Entwicklungsumgebung

G. Komponenten-/Klassendiagramme der Blockchain-Anwendungen

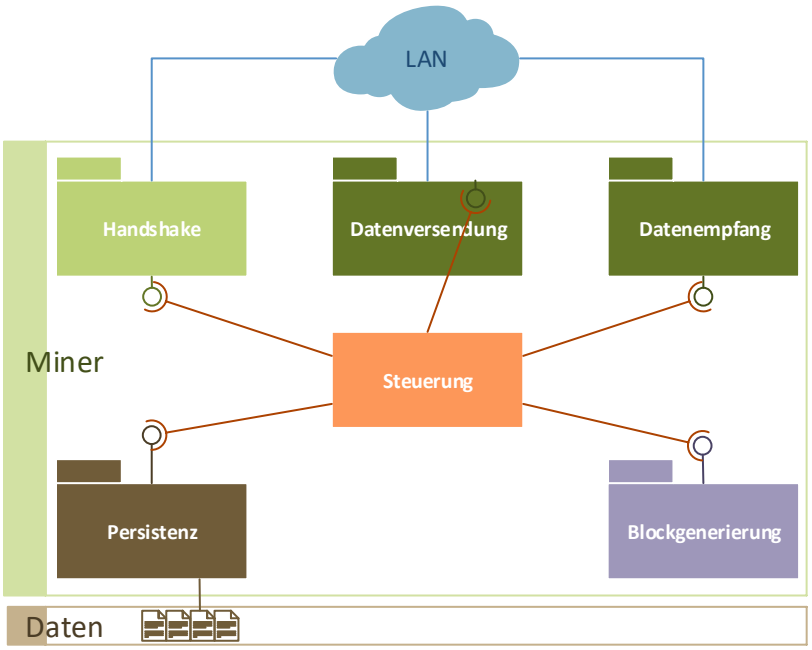

Abbildung 34: Komponenten-Diagramm des Miners

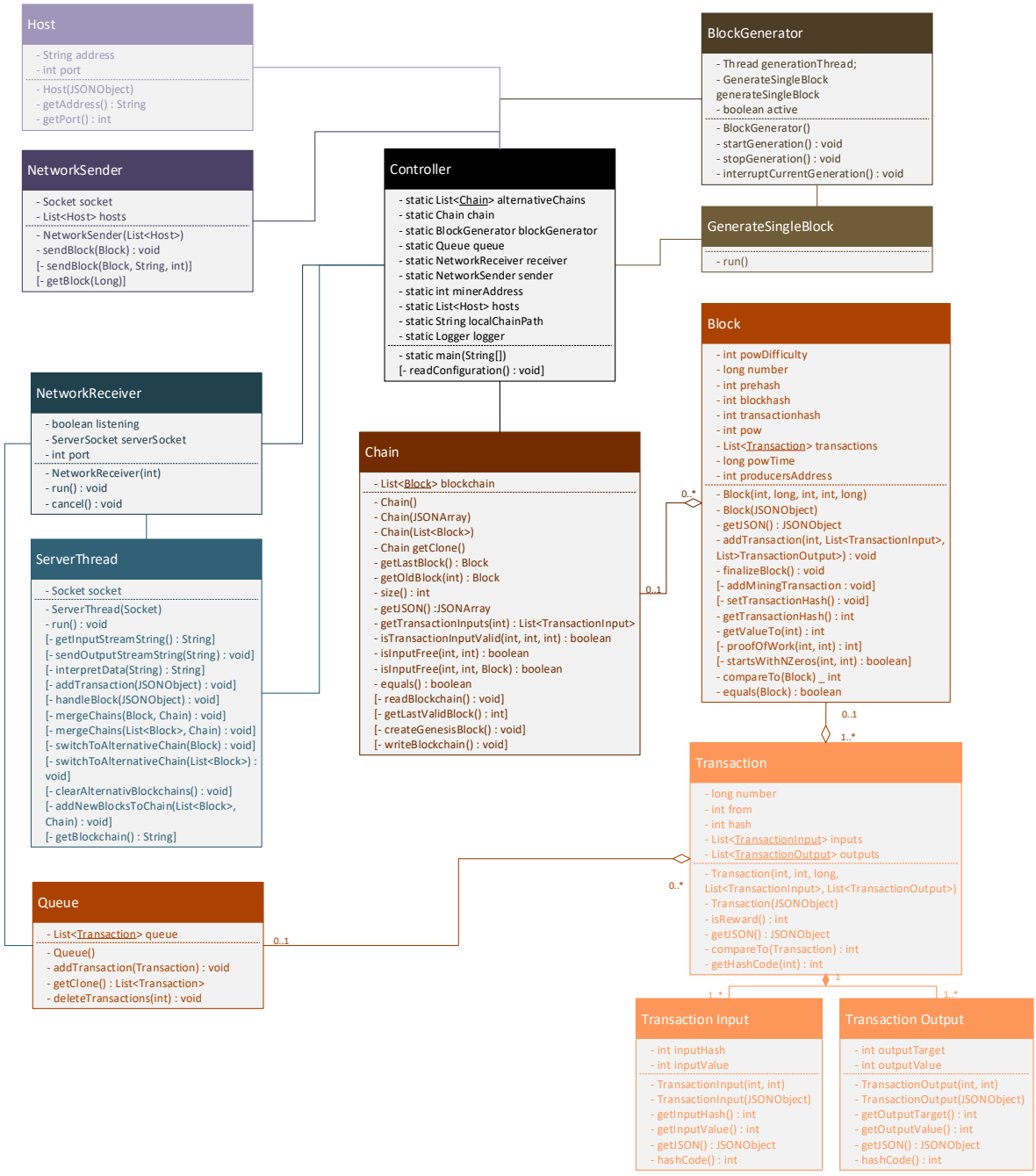

Abbildung 35: Klassendiagramm Mininganwendung

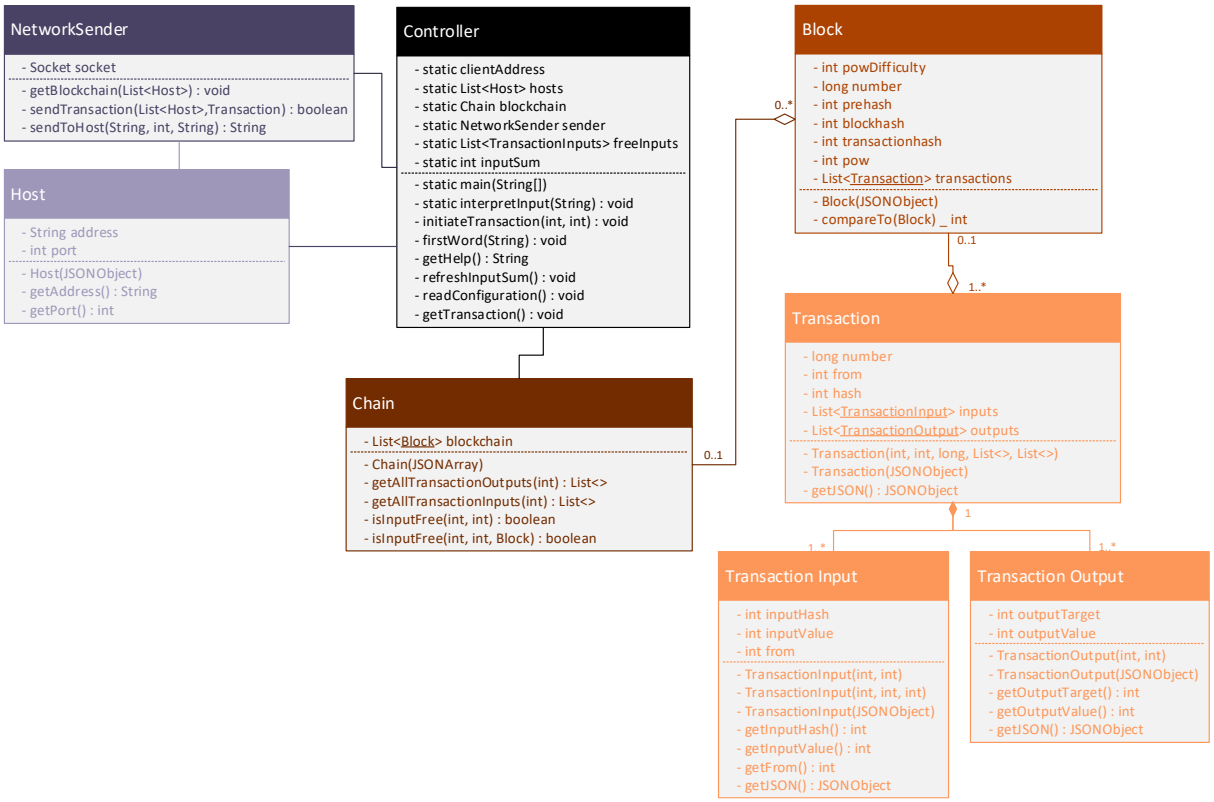

Abbildung 36: Klassendiagramm der Wallet-Anwendung

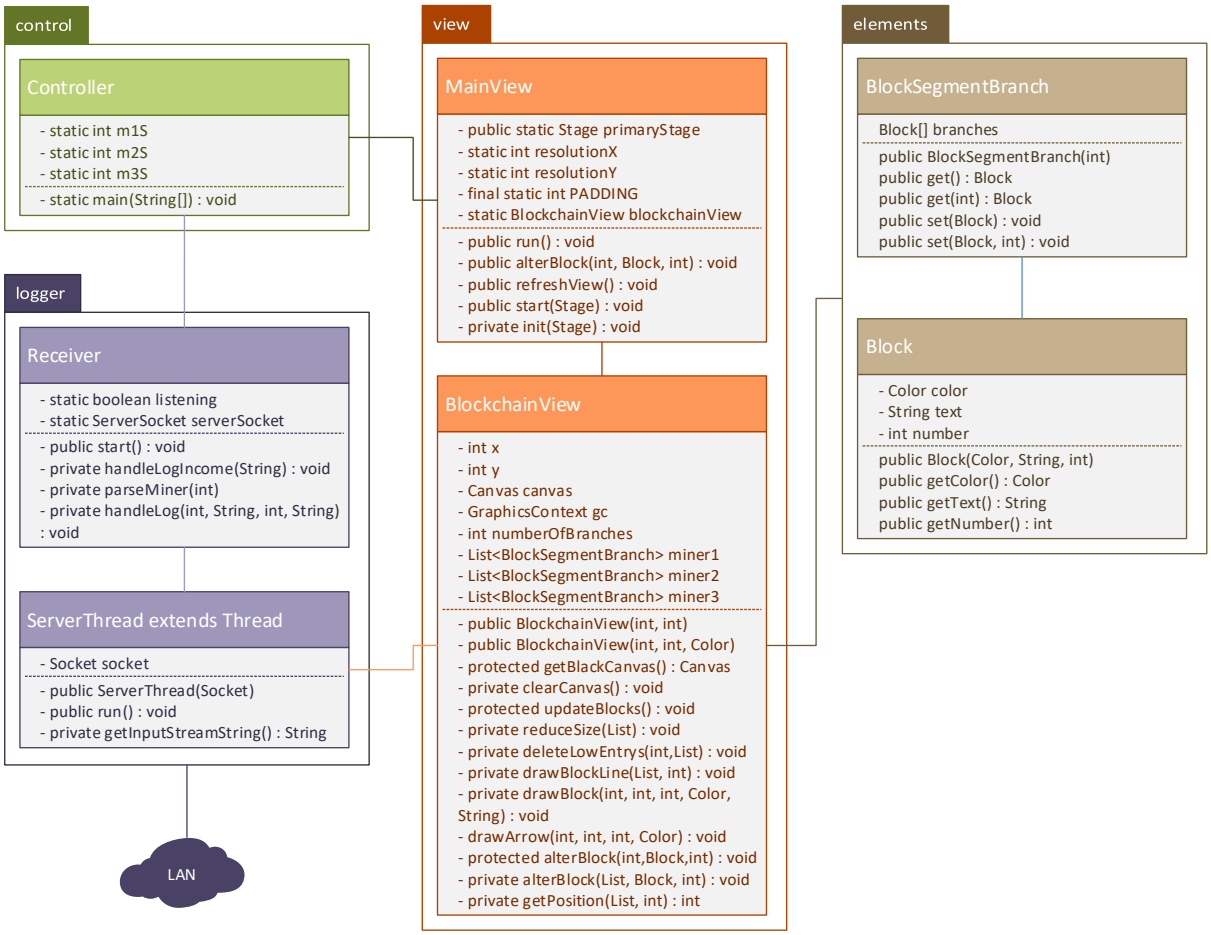

Abbildung 37: Klassendiagramm Visualisierungsanwendung

H. Blockgenerierung der Mining-Anwendungen

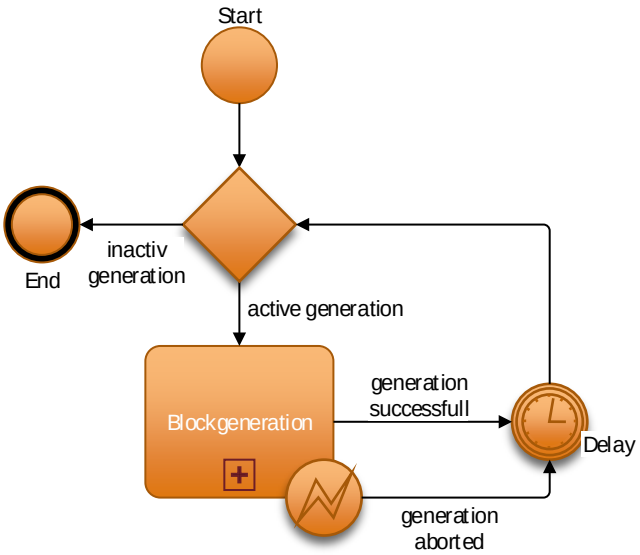

Abbildung 38: Blockgenerierung Übersicht

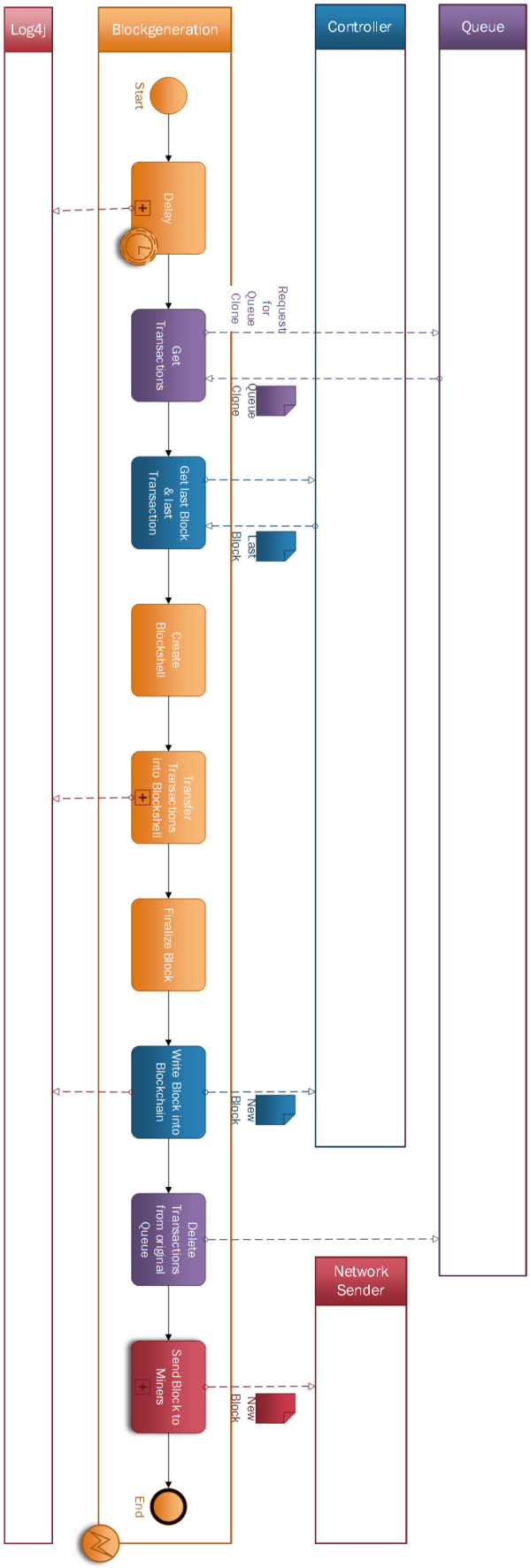

Abbildung 39: Blockgenerierung Detailansicht

I. Angriff auf die prototypische Blockchain

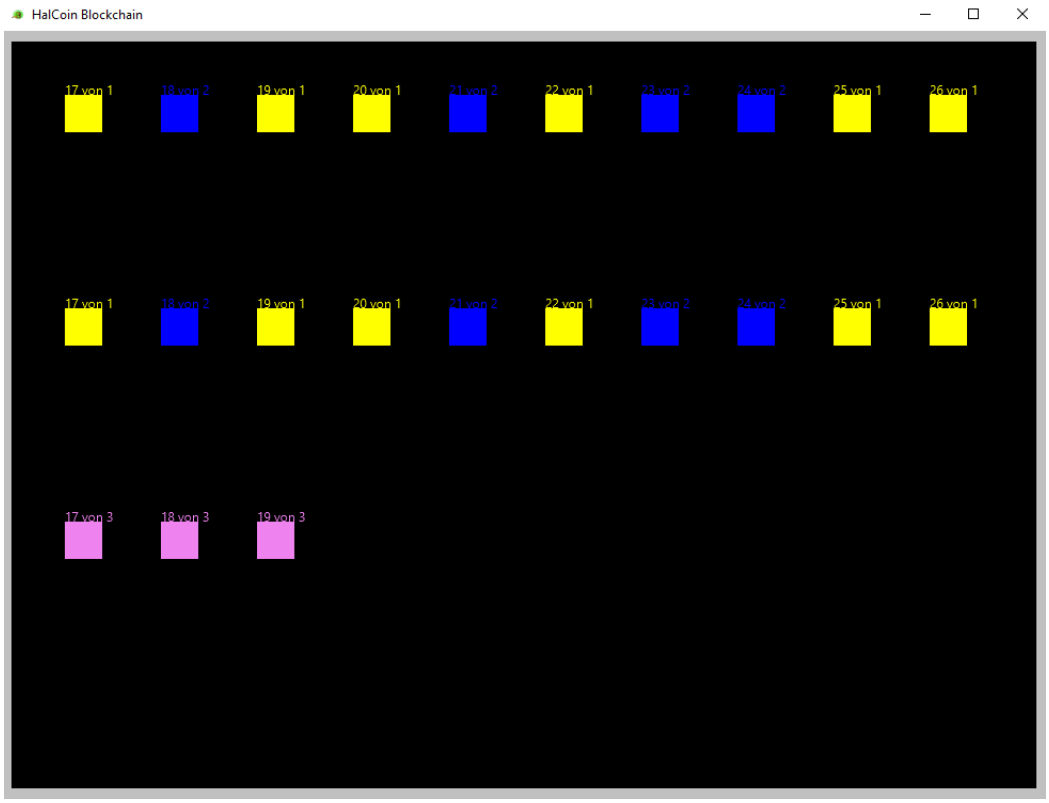

Abbildung 40: Angriff auf den Prototyp - Erscheinen des Angreifers

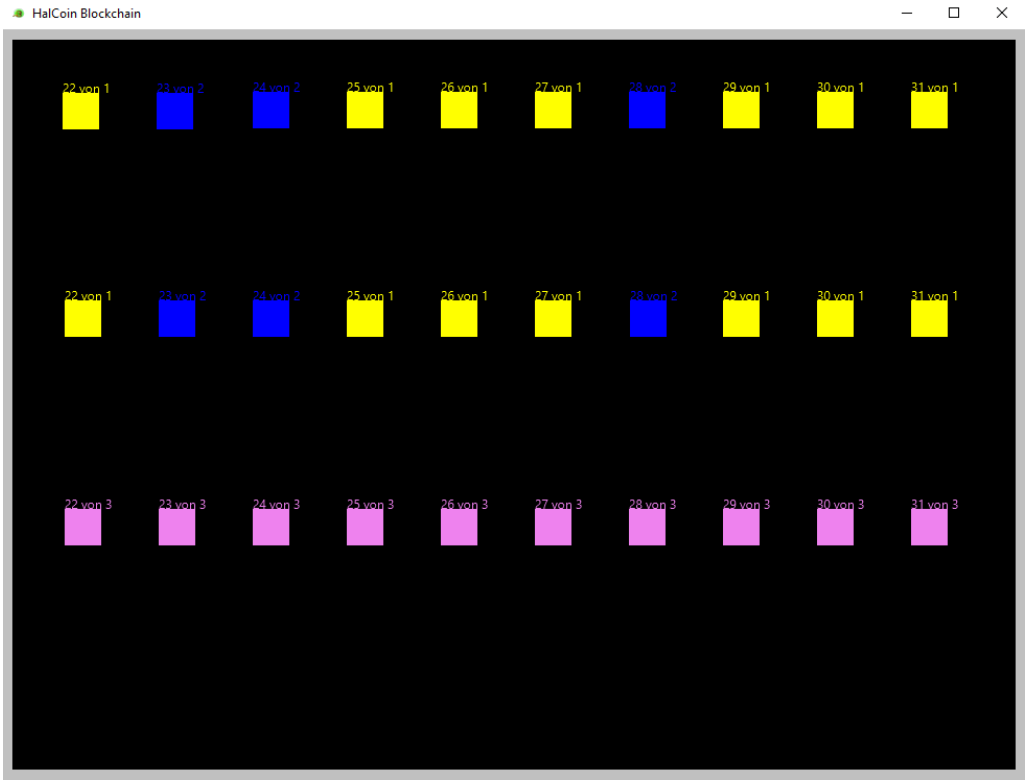

Abbildung 41: Angriff auf den Prototyp - Angreifer ist gleichauf

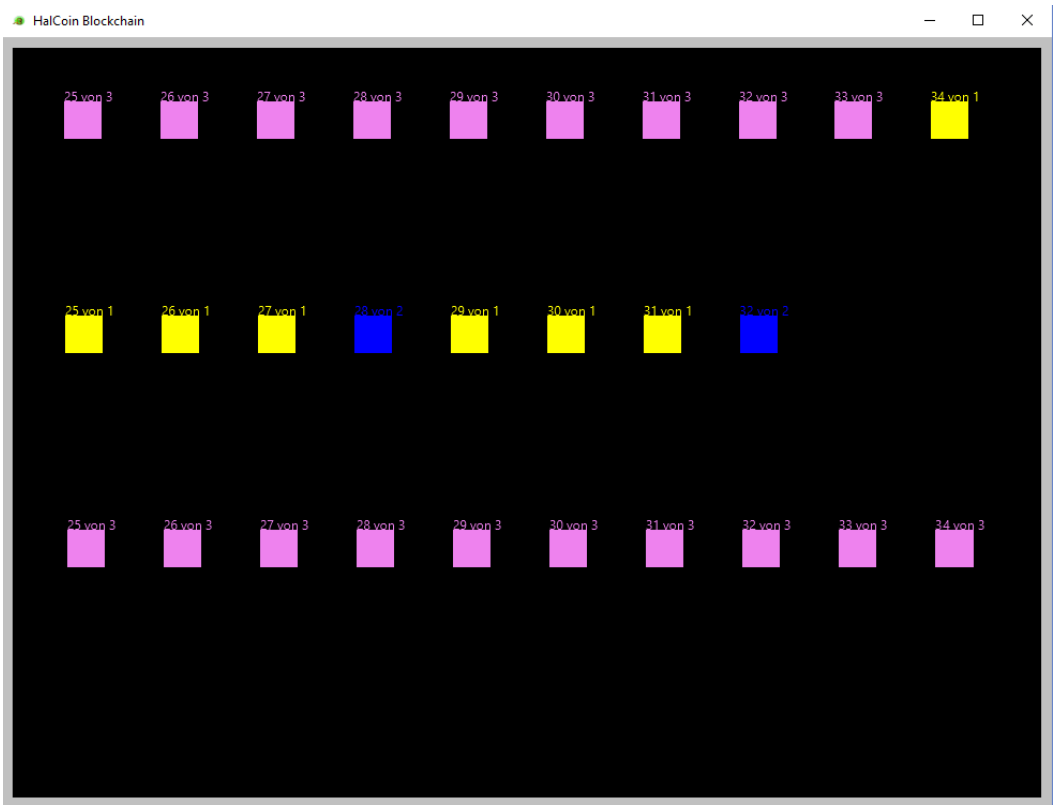

Abbildung 42: Angriff auf den Prototyp - Angreifer beeinflusst Miner 1

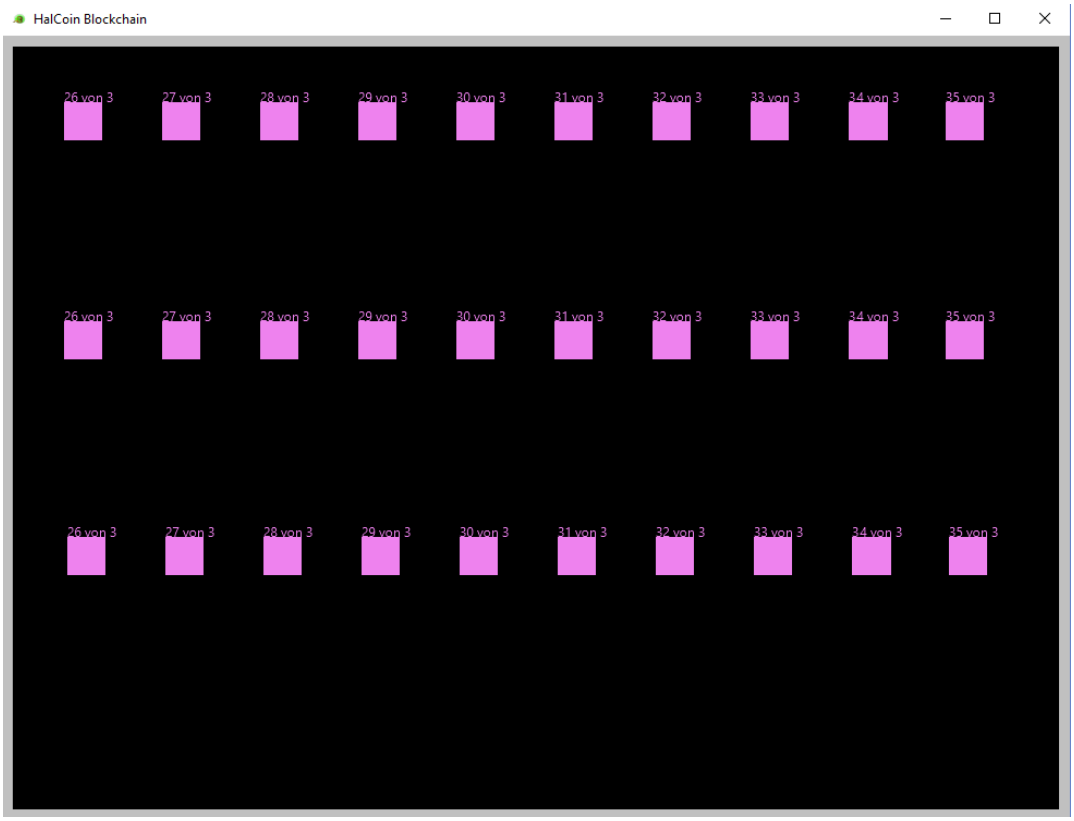

Abbildung 43: Angriff auf den Prototyp - Angreifer beeinflusst Miner 2

J. (Kompilierter) Quellcode der Blockchain Anwendungen

Der Quellcode der Mining-Anwendung liegt als zip-Archiv „mining_code.zip“, der Quellcode der Visualisierungsanwendung als zip-Archiv „visualisierung_code.zip“ und die Wallet-Anwendung als zip-Archiv „wallet_code.zip“ vor.

Zudem liegen die kompilierten Anwendungen samt Konfigurationsanleitung (readme.txt) als zip-Archiv „blockchain-kompilate.zip“ vor.

K. Tabellarischer Vergleich der Frameworks

	Meteor	**Embark**	**Truffle**
Typ	Full Stack Framework, JavaScript App Plattform	framework that allows you to easily develop and deploy Ethereum DAPPs	Development environment, testing framework and asset pipeline for Ethereum (DAPPs)
Arten	generisch für DAPPs	Fokus von DAPPs auf Ethereum	Fokus von DAPPs auf Ethereum
Funktionsumfang			
Kompilieren/Deployment von Smart Contracts? (zzl. Packages, Out-of-the-Box)	intern per web3 Object (über Ethereum Packages) oder extern (per geth oder Packages (embark))	integriert (alle .sol - Dateien im Contract Ordner werden deployt mit einem Befehl)	integriert, nutzt als Abstraktion Pudding (https://github.com/ConsenSys/ether-pudding)
Testing?		Unterstützt Test-Driven Development von Contracts über eigene JS Lib, für JS Default: Jasmine, andere möglich	Mocha (Tests) Chai (Assertions), TestRPC Client für automated Testing
Deployment/Veröffentlichung DAPP?	Unbundle von Code oder Bundling als Meteor Client (Packages: meteor-build-client)	Unbundled	distributable version of app (minified)
Debugging von Smart Contracts/Code?	keine DIE	keine DIE	keine DIE
Deployment Testnet?	geth-abhängig, theoretisch auch TestRPC Client möglich	integriert per Befehl, Spezifikation in Datei, theoretisch TestRPC Client möglich	mittels TestRPC Client, Spezifikation in Datei

Unterstützte Sprachen für Smart Contract?	Solidity	Solidity, (Serpant)	Solidity
API?	Ethereum Web3 (über Packages)	Ethereum Web3 integriert	Pudding (Abstraktion für Web3)
Handhabung			
Live Reloading?*	integriert	integriert	integriert
Generalisierung Smart Contracts?	extern, out-of-the-box nicht möglich	integriert (Ablage in Verzeichnis, deploy per Befehl)	integriert
Integration von Artefakten (JS)?	über Meteor-Packages oder im Code	im Code/Ordner	im Code/Order
Struktur von DAPP	ziemlich ähnlich, bis auf, dass Embark und Truffle ein Verzeichnis haben, wo Smart Contracts abgelegt werden, die automatisch deployt werden können		
Reifegrad/Verbreitung			
Nutzer?	Ethereum-Dev	EthereumJ-Dev	-
Version?	1.2.1	1.0.2	0.0.4
Erweiterbarkeit?	über Meteor -Packages (z.B. Ethereum web3 Objekt)	per Javascript-Libs im Code, Ethereum integriert	per Javascript-Libs im Code, Ethereum integriert
Entwicklungsaktivität/ Wartung/Weiterentwicklung			
Commiter?	249	14	6
Letzter Commit?	04.12.2015	06.12.2015	04.12.2015
Road Map?			
Unternehmen?	Meteor Development Group	-	ConsenSys
Open Source?	ja	ja	ja, MIT
Abhängigkeiten zu			
Installation? (z.B. NPM?)	-	geth (1.1.0), solc (0.1.0) or serpent	

		(develop), node (0.12.2) and npm	
Deployment DAPP?	-	-	-
Interaktion/Deployment Ethereum?	mittels RPC über z.B. geth	mittels RPC über z.B. geth	mittels RPC über z.B. geth
Entwicklungsprozess			
(Default) Workflow? (Einfachheit)	Per Texteditor, Befehle Kommandozeile	Per Texteditor, Befehle Kommandozeile	Per Texteditor, Befehle Kommandozeile
Umfang	Develop, Deploy, Test	Develop, Deploy, Test	Develop, Deploy, Test
Toolunterstützung (z.B. IDE)?	-	-	-
Deploymentprozess			
Workflow? (Einfachheit) / Umfang?	Smart Contracts über Geth oder per Package (Embark)	integriert, per Befehl	integriert, per Befehl
Vorbereitungen			
Toolunterstützung (z.B. IDE)?	-	-	-
Ausführung von Artefakten(DAPP, Client)			
Bundling in Client?	möglich	-	-
Vorraussetzungen?	Installation von Meteor als Ausführungsumgebung	Installation Embark	Installation Truffle
Besonderheit	https://github.com/hitchcott/meteor-embark, Integration	RPC Simulator	Abstraktions-Layer mit Pudding für Interaktion mit Smart Contracts

	von Embark Framework als Package in Meteor		
	Doku	Ausführliches Wiki	Wiki
*Änderungen werden erkannt und DAPP neu geladen			
Test RPC Client (pyethereum Basis)	https://github.com/ConsenSys/eth-testrpc		

Tabelle 6: Vergleich der Frameworks

L. Technischer Basis-Workflow der DAPP

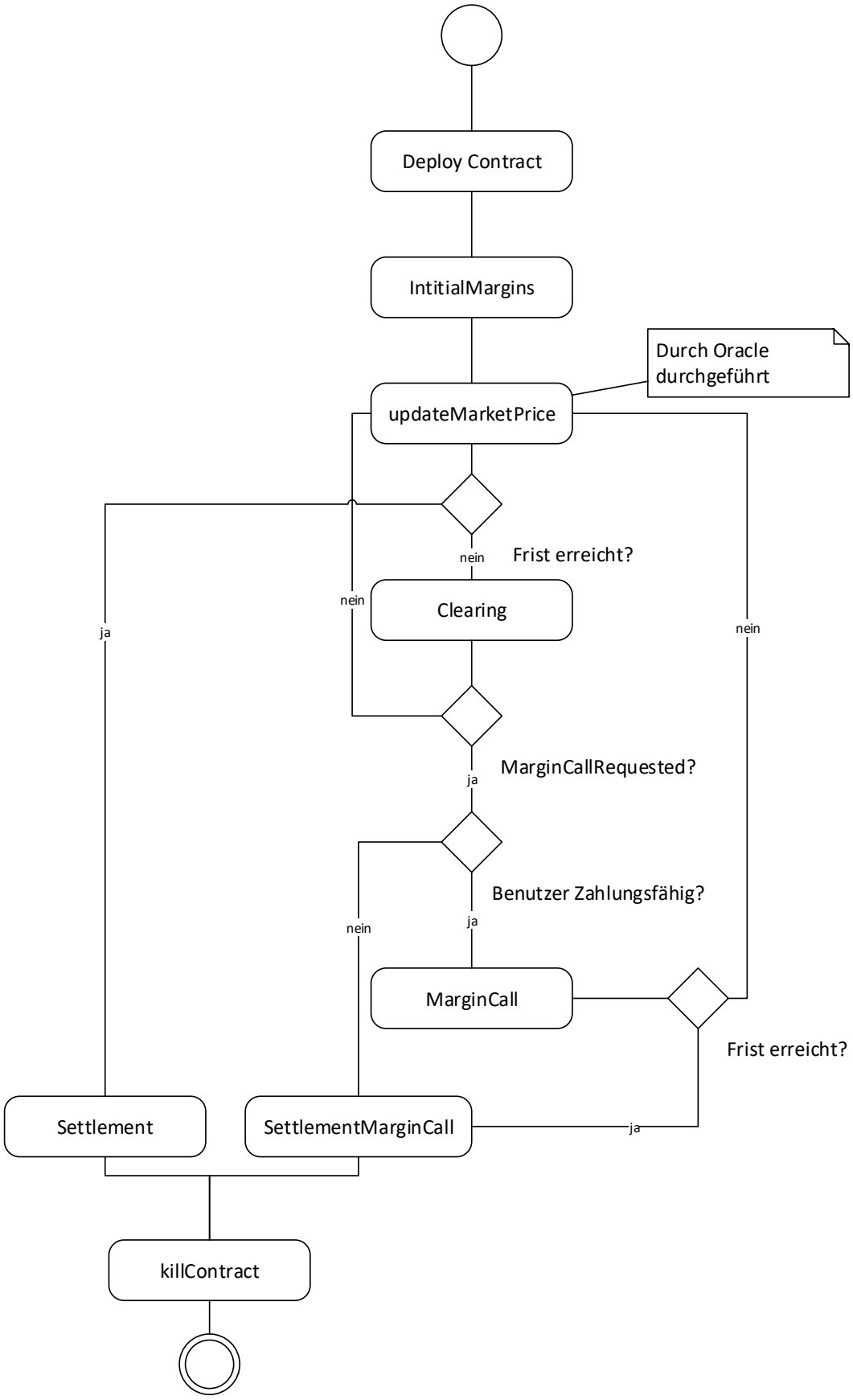

Abbildung 44: Basis-Workflow Clearinghouse DAPP

M. Quellcode der Clearinghouse-DAPP

Der Quellcode der Clearinghouse-DAPP liegt als zip-Archive „clearinghouse_dapp_artefakte_ct.zip“ vor.

N. BPMN-Diagramm des Geschäftsvorfalls

Das BPMN-Diagramm ist aufgrund seiner Größe als Anlage zu dieser Arbeit in der Datei „BPMN – Geschäftsvorfall.pdf“ zu finden.

O. BPMN-Diagramm des Unterprozess prüfe Kontostände

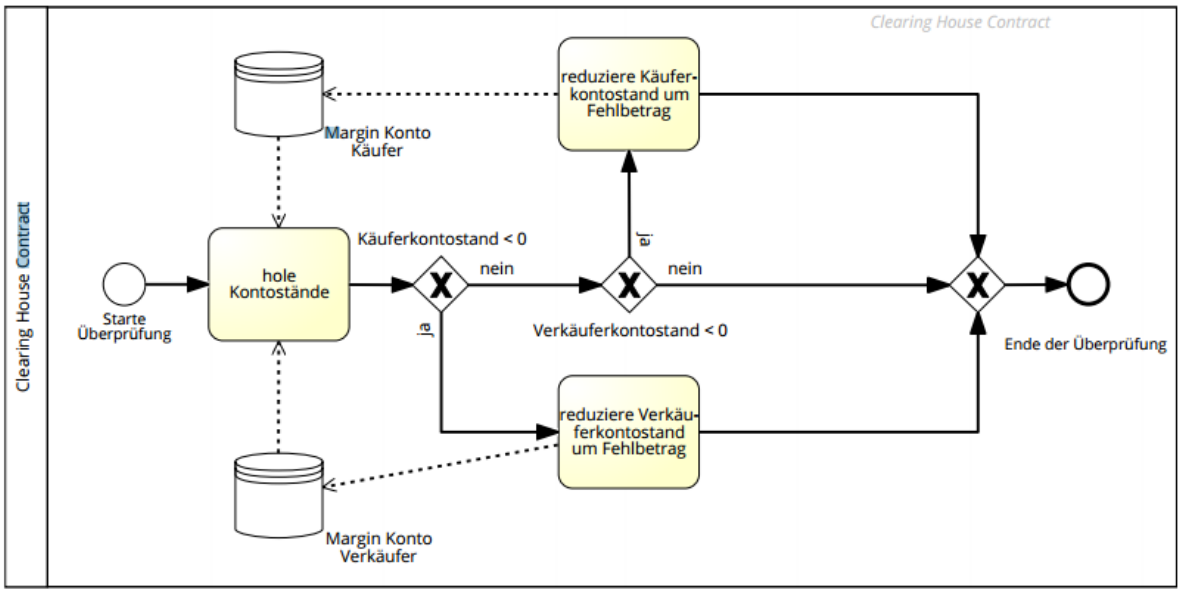

Abbildung 45: Unterprozess prüfe Kontostände

P. BPMN-Diagramm des Smart Oracles

Das BPMN-Diagramm ist aufgrund seiner Größe als Anlage zu dieser Arbeit in der Datei „BPMN – Smart Oracle.pdf“ zu finden.

Q. Quellcode des Smart Oracles

Der Quellcode befindet sich in der Datei „Oracle.zip“