

Masterarbeit

Einsatz und Vergleich verschiedener Blockchain-Technologien am Beispiel einer Glücksspielanwendung

Eingereicht von:
Dany BROSSEL
Matrikelnummer: 3024062

Studiengang: Information Systems Engineering

4. Juni 2018

Betreuer: **Prof. Dr. rer. nat. Dr.-Ing. Georg HOEVER**
Korreferent: **Prof. Dr. Marco SCHUBA**

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Datum:

Unterschrift:

FH AACHEN

Zusammenfassung

Einsatz und Vergleich verschiedener Blockchain-Technologien am Beispiel einer Glücksspielanwendung

von Dany BROSEL

Diese Ausarbeitung zeigt die Interaktion mit dem Bitcoin und Ethereum Netzwerk am Beispiel einer Glücksspielanwendung. Die Integration in die Anwendung erfolgte im Falle von Bitcoin über einen Simple Payment Verification Node. Im Falle von Ethereum wird die Integration über einen Full (validating) Node demonstriert. Durch die Verwendung von Smart Contracts wird für diesen Anwendungsfall vollständig auf den Einsatz einer vertrauenswürdigen Drittpartei (Trusted Third Party) verzichtet.

Inhaltsverzeichnis

Zusammenfassung	ii
1 Einleitung	1
1.1 Motivation	1
1.2 Ziel	1
1.3 Projektidee	2
1.4 Anforderungen	2
1.5 Vorhandenes	3
1.5.1 Cyberdice Protokoll	3
1.5.2 Glücksspielseiten	3
1.6 Aufbau dieser Arbeit	3
2 Erster Ansatz: Bitcoin	4
2.1 Grundlagen	4
2.1.1 Peer-to-Peer Netzwerk	4
2.1.2 Blockchain	5
2.1.3 Konsensregeln	7
2.1.4 Proof-of-Work und Mining	7
2.2 Konzept	9
2.3 Umsetzung	15
2.3.1 Interaktion mit dem Bitcoin Netzwerk	15
2.3.2 Überblick	17
2.3.3 Datenmodell	18
2.3.4 Geschäftslogik	18
2.3.5 Grafische Benutzeroberfläche	26
2.4 Evaluation	31
2.4.1 Prüfung der Anforderungen	31
2.4.2 Gewinnerauswahl	32
2.4.3 Verteilung der Blockhash-Werte	33
2.4.4 Manipulationsversuch durch Miner	34
2.4.5 Auszahlungstransaktion	37
2.4.6 Blockchain Mining Varianz	39
2.4.7 Blockchain Forks	40
2.4.8 Betrugsmöglichkeiten	40
3 Zweiter Ansatz: Ethereum	41
3.1 Grundlagen	41
3.1.1 Smart Contracts	41
3.1.2 Ethereum Accounts	41
3.1.3 Transaktionen	42
3.1.4 Nachrichten	42
3.1.5 Ether	42
3.1.6 Ethereum Virtual Machine	43

3.1.7	Systemzustand Übergangsfunktion	43
3.1.8	Systemzustand Beispiel	44
3.1.9	Unterschiede zu Bitcoin	45
3.2	Konzept	47
3.3	Umsetzung	48
3.3.1	Überblick	48
3.3.2	Smart Contract	48
3.3.3	Smart Contract Bereitstellung	51
3.3.4	Geschäftslogik Glücksspielanwendung	54
3.3.5	Grafische Benutzeroberfläche	59
3.4	Evaluation	65
3.4.1	Prüfung der Anforderungen	65
3.4.2	Aufruf der Auszahlungstransaktion	66
3.4.3	Verteilung der Hashfunktion Keccak-256	67
3.4.4	Sicherheit von Smart Contracts	67
4	Sonstige Distributed-Ledger-Technologie	69
4.1	Directed Acyclic Graph	69
4.2	Proof of Stake	70
4.3	Second Layer Solutions	70
4.3.1	Payment Channel	70
4.3.2	Lightning Netzwerk	71
5	Fazit	72
A	Vorhandene Glücksspielseiten	73
A.1	Bitcoin	73
A.2	Ethereum	75
	Quellenverzeichnis	76

Abkürzungsverzeichnis

ECDSA	E lliptic C urve D igital S ignature A lgorithm
ASIC	A pplication S pecific I ntegrated C ircuit
BIP	B itcoin I mprovement P roposal
GUI	G raphical U ser I nterface
RPC	R emote P rocedure C all
EJB	E nterprise J ava B eans
EIP	E thereum I mprovement P roposal
ABI	A pplication B inary I nterface
JSON	J ava S cript O bject N otation
DAG	D irected A cyclic G raph
SPV	S implified P ayment V erification
JVM	J ava V irtual M achine

Kapitel 1

Einleitung

Die Erfindung der Kryptowährung Bitcoin und deren inhärente Blockchain-Technologie hat in den letzten Jahren einen regelrechten Hype ausgelöst. Begriffe wie Blockchain, Distributed Ledger und Smart Contract sind das bestimmende Thema schlechthin. Blockchain-Technologie verspricht, durch den Einsatz von Dezentralität, Unveränderlichkeit und Transparenz, die Möglichkeit zwischen anonymen, sich gegenseitig nicht trauenden Parteien, digitale Werte auszutauschen.

1.1 Motivation

Das Internet vernetzt weltweit mehrere Millionen Menschen. Es ermöglicht diesen anonym zu kommunizieren und miteinander zu interagieren. Aufgrund der Anonymität und des damit einhergehenden Betrugspotentials benötigen die meisten Anwendungsfälle den Einsatz einer sogenannten Trusted Third Party als Mittelsmann. Das fehlende Vertrauen zwischen den anonym agierenden Parteien wird durch diese dritte, nicht anonym auftretende, Partei kompensiert. Diese nicht anonyme Partei muss sich den länderspezifischen Regeln und Gesetzen unterwerfen und kann ihren Service daher nur in einem gewissen, vorgegebenen Rahmen betreiben. Da das Betreiben eines solchen Services mit gewissen Kosten verbunden ist, werden diese meist auf die Nutzer abgewälzt.

Online Casinos sind ein Beispiel für eine solche Trusted Third Party. Sie bieten eine Plattform, die Spieler vernetzt und vor gegenseitigem Betrug schützt. Außerdem verwalten Online Casinos das Geld ihrer Spieler. Die Spieler vertrauen den Casinos diese Aufgabe an, da es sich um registrierte Firmen handelt, die juristisch haftbar gemacht werden können. Ein weiterer Aspekt bei dem Vertrauen eine Rolle spielt sind die angebotenen Spiele selbst. Egal ob Black Jack, Poker oder Roulette, die meisten solcher Spiele erfordern die Generierung von Zufallszahlen, beispielsweise wenn Spielkarten verteilt werden. Die Spieler haben dabei keine Möglichkeit nachzuprüfen, ob der Algorithmus, der ihnen die Karten zuteilt, auch wirklich fair ist. Die Spieler müssen dem Casino somit vertrauen, dass dieses sie nicht benachteiligt.

1.2 Ziel

Ziel dieser Masterarbeit ist es den Einsatz von Blockchain-Technologie an der beispielhaften Realisierung einer Glücksspielanwendung zu demonstrieren. Der Einsatz einer Blockchain soll dabei das Vertrauen, das der Endnutzer der Anwendung entgegenbringen muss, auf ein Minimum reduzieren. Falls möglich soll vollständig auf den Einsatz einer Trusted Third Party verzichtet werden.

1.3 Projektidee

Die in dieser Arbeit betrachtete Glücksspielanwendung soll ein Spiel anbieten, bei dem Teilnehmer in einen Geldtopf einzahlen und anschließend auf ein zufälliges Ereignis wetten. Jeder Teilnehmer soll dabei die gleichen Gewinnchancen haben. Sobald alle Teilnehmer eingezahlt haben, wird einer der Teilnehmer zufällig ausgewählt und gewinnt den gesamten Geldtopf. Der Gewinner bekommt somit seinen eigenen Einsatz als auch den Einsatz aller Mitspieler ausgezahlt. Die restlichen Teilnehmer verlieren und gehen leer aus.

Die erstmalig in Bitcoin verwendete Blockchain-Technologie ist für die Entwicklung einer solchen Anwendung bestens geeignet, da sie transparente, pseudonyme Zahlungen ermöglicht. Außerdem lässt sich der für die Gewinnerauswahl benötigte Zufall durch einen in der Zukunft liegenden Zustand der Blockchain ableiten. Der Zufallsfaktor kommt somit direkt von der Blockchain und ist für alle Teilnehmer nachvollziehbar.

1.4 Anforderungen

Die zu realisierende Glücksspielanwendung muss den folgenden Anforderungen gerecht werden.

1) Transparente Einzahlungen

Die Einzahlung jedes Endnutzers ist für jeden anderen Endnutzer nachprüfbar.

2) Gewinnerauswahl durch Zufallsfaktor

Die Auswahl des Gewinners ist von einem zufälligen Faktor abhängig, auf den weder die Anwendung noch die Endnutzer einen Einfluss haben.

3) Nachprüfbarkeit des Zufallsfaktors

Jeder Endnutzer kann die Echtheit des zufälligen Faktors eigenständig nachprüfen.

4) Transparente Auszahlungen

Die Auszahlung an den Gewinner muss transparent und somit für jeden Endnutzer nachprüfbar sein.

5) Fairheit des Spiels

Jeder Endnutzer besitzt die gleiche Gewinnwahrscheinlichkeit und niemand wird benachteiligt.

1.5 Vorhandenes

1.5.1 Cyberdice Protokoll

Einen ersten Ansatz wie man im Internet Glücksspiel ohne eine vertrauenswürdige Drittpartei betreiben kann, liefert [35]. Es stellt ein Kommunikationsprotokoll vor, das mit Hilfe kryptographischer Methoden sicherstellt, dass weder die Teilnehmer noch Außenstehende betrügen können. Das zum Glücksspiel verwendete Protokoll funktioniert aber nur unter der Annahme, dass es eine zentrale Institution (Bank) gibt, bei der die Teilnehmer Geld einzahlen und im Falle eines Gewinns, gegen die Vorlage eines Beweises, Geld ausgezahlt bekommen. Durch die Erfindung dezentraler Kryptowährungen, die auf einer für jeden einsehbaren Blockchain basieren, fällt diese vorher noch benötigte zentrale Institution weg.

1.5.2 Glücksspielseiten

Es gibt bereits Services die dezentrales, transparentes Glücksspiel mit Hilfe von Kryptowährungen umsetzen. Die Internetseite [10] bietet diverse Spiele an, bei denen die Nutzer mit Kryptowährungen bezahlen können. Die Internetseite [39] bietet ein Würfelspiel an, das durch einen Smart Contract auf der Ethereum Plattform umgesetzt ist. Eine Beschreibung der verwendeten Verfahren zur Gewinnerauswahl befindet sich im Anhang dieser Ausarbeitung.

1.6 Aufbau dieser Arbeit

In dieser Arbeit wird zunächst Bitcoin als Beispiel einer auf Blockchain-Technologie basierenden Kryptowährung betrachtet. Anschließend wird der Einsatz von Smart Contracts mit Ethereum präsentiert. In beiden Fällen werden zunächst die relevanten Grundlagen geklärt und ein Konzept vorgestellt. Anschließend wird dieses als Glücksspielanwendung mithilfe der jeweiligen Technologie umgesetzt. Die beiden resultierenden Glücksspielanwendungen werden letztendlich unter Zuhilfenahme der aufgestellten Anforderungen evaluiert. Nach diesem Hauptteil werden abschließend weitere Blockchain-Technologien und deren Anwendbarkeit für die Projektidee betrachtet.

Kapitel 2

Erster Ansatz: Bitcoin

2.1 Grundlagen

Bei Bitcoin handelt es sich um die erste digitale, dezentral organisierte Währung. Die Idee digitaler Währungen existiert bereits seit der Erfindung des Internets. Allerdings scheiterten diese in der Vergangenheit daran, dass sie auf einen zentralen Punkt der Kontrolle angewiesen waren und somit einen Single Point of Failure beinhalteten. Die am 03. Januar 2009 gestartete digitale Währung "Bitcoin" schaffte es erstmalig gänzlich auf die Verwendung einer zentralen Instanz zu verzichten und somit ein verteiltes, dezentrales und sicheres digitales Zahlungssystem zu realisieren. Bitcoin wurde in dem vom Pseudonym "Satoshi Nakamoto" veröffentlichten Paper [28] "Bitcoin: A Peer-to-Peer Electronic Cash System" das erste Mal beschrieben. Bitcoin funktioniert durch das Zusammenspiel mehrerer Komponenten und besteht aus:

- Einem dezentralen *Peer-to-Peer Netzwerk*, das mit Hilfe des Bitcoin-Protokolls kommuniziert.
- Der *Blockchain*, die eine öffentliche Transaktionsdatenbank darstellt, die alle validen Transaktionen seit dem Start des Netzwerks aufzeichnet.
- Einer Menge an *Konsensregeln* mit Hilfe denen Netzwerkteilnehmer eigenständig Transaktionen auf ihre Gültigkeit prüfen können.
- Einem *Proof-of-Work Algorithmus*, der es den Teilnehmern erlaubt, sich in dem globalen dezentralen Netzwerk auf den Zustand der Transaktionsdatenbank zu einigen. Das kontinuierliche Ausführen des Proof-of-Work Algorithmus wird *Mining* genannt.

2.1.1 Peer-to-Peer Netzwerk

Peer-to-Peer-Netzwerke sind Netzwerke, die auf direkten Verbindungen zwischen Rechnern beruhen, ohne dass dabei einer der Rechner eine Sonderstellung einnimmt oder ein Server die Kommunikation vermittelt. In einem reinen Peer-to-Peer-Netzwerk sind alle Computer gleichberechtigt und können sowohl Dienste in Anspruch nehmen, als auch zur Verfügung stellen. Das Peer-to-Peer-Modell ist somit grundlegend verschieden von dem im Internet am häufigsten verwendeten Client-Server-Modell. Da jeder Knoten des Netzwerks gleichzeitig Client und Server ist, gibt es keine zentrale Instanz, die einen sogenannten *Single Point of Failure* darstellt.

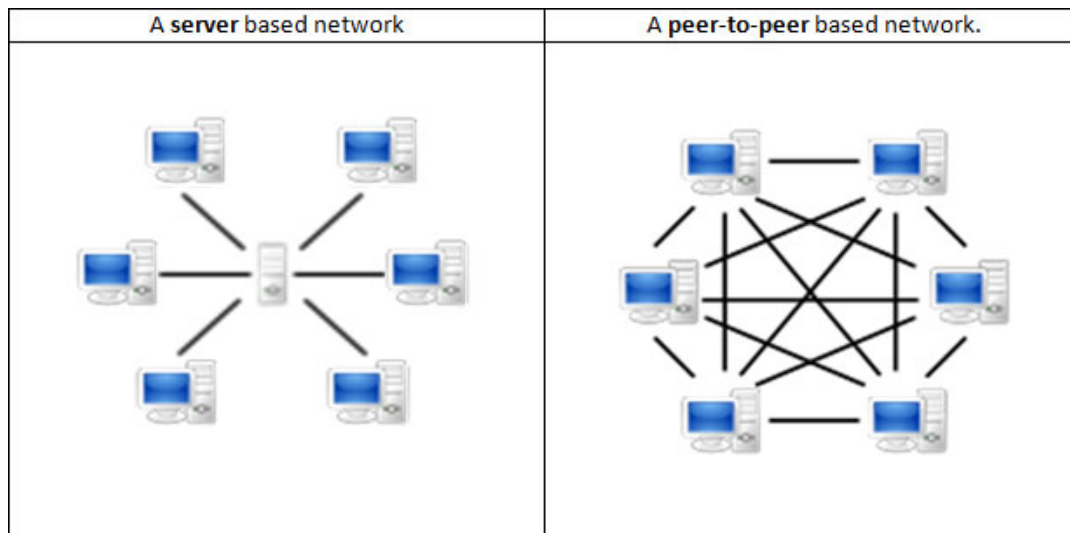


ABBILDUNG 2.1: Client-Server | Peer-to-Peer [31]

Peer-to-Peer Netzwerke sind selbstorganisierend. Das Hinzufügen neuer Teilnehmer und das Entfernen bestehender Netzwerkknoten findet ohne eine zentrale Verwaltung statt und behindert die Funktionsweise des Netzwerks nicht. Jeder Knoten des Netzwerks verwaltet eigenständig seine direkten Nachbarknoten. Die Art und Weise wie die Teilnehmer des Netzwerks miteinander kommunizieren ist durch das Netzwerkprotokoll vorgegeben. Um am Netzwerk teilzunehmen braucht man nur eine Software, die das Netzwerkprotokoll implementiert und einen Internetanschluss. Im Falle der Kryptowährungen nennt man diese Software "Wallet" (englisch für Brieftasche), da man mit ihr Zahlungen initiieren und empfangen kann. Möchte ein Teilnehmer Bitcoins an einen anderen Teilnehmer senden, erstellt er dazu eine Nachricht, die solch eine Transaktion beinhaltet und schickt sie an seine direkten Nachbarn des Peer-to-Peer Netzwerks. Die Nachbarn prüfen die Gültigkeit der Transaktion und leiten diese gegebenenfalls an ihre Nachbarn weiter. Auf diese Art und Weise verteilt sich die Transaktion im gesamten Netzwerk.

2.1.2 Blockchain

Eine Blockchain ist eine global verteilte Transaktionsdatenbank. Jeder Teilnehmer des Peer-to-Peer Netzwerkes kann lokal eine Kopie dieser Datenbank speichern. Dies erlaubt es ihm jegliche Datenbankeinträge zu lesen. Im Gegensatz zum lesen den Zugriff ist der schreibende Zugriff auf die Datenbank nur unter sehr strikten Regeln möglich. Über diese Regeln sind sich alle Teilnehmer des Netzwerkes einig. Daher werden diese Regeln Konsensregeln genannt. Möchte ein Teilnehmer eine Transaktion in die Datenbank schreiben, muss er sicherstellen, dass sie den Konsensregeln entspricht. Falls die Transaktion eine Konsensregel bricht, wird sie vom Netzwerk verworfen und es ist ausgeschlossen, dass sie in die Blockchain aufgenommen wird. Eine Transaktion beschreibt den Übergang von einem alten Systemzustand in einen neuen Systemzustand. Im Fall von Bitcoin handelt es sich bei dem Systemzustand um ein digitales Kontenbuch. Die Konten sind bei Bitcoin sogenannte Adressen und repräsentieren den öffentlichen Schlüssel eines ECDSA Schlüsselpaars. Um die einer Adresse zugeschriebenen Bitcoins zu überweisen, muss der Besitzer mit Hilfe des privaten Schlüssels eine digitale Signatur erstellen. Diese Signatur garantiert, dass die Überweisung vom rechtmäßigen Besitzer der Bitcoins autorisiert ist.

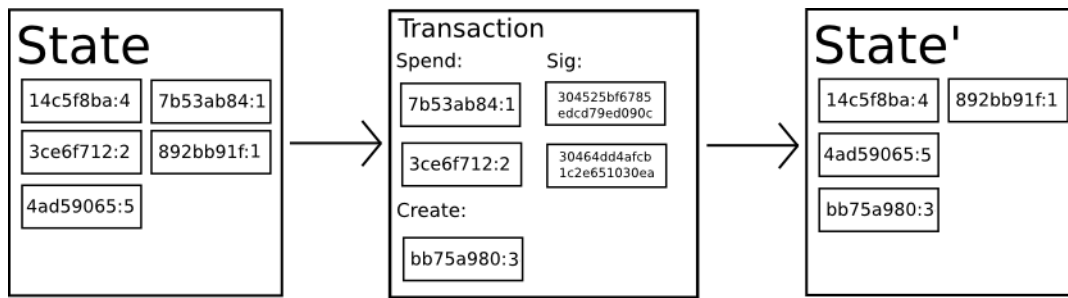


ABBILDUNG 2.2: Bitcoin Zustandsveränderung durch Transaktion [6]

Die Transaktion aus Abbildung 2.2 überweist 1 Bitcoin von der Adresse 7b53ab84 und 2 Bitcoin von Adresse 3ce6f712 auf die Adresse bb75a980 und überführt somit das Kontobuch in einen neuen Zustand. Möchten mehrere Teilnehmer den Systemzustand durch Transaktionen gleichzeitig anpassen, spielt die Reihenfolge, in der die Transaktionen ausgeführt werden, eine wichtige Rolle. Aus diesem Grund werden Transaktionen in sogenannten Blöcken, in einer festen Reihenfolge, aggregiert. Somit werden nicht einzelne Transaktionen, sondern ganze Blöcke von Transaktionen in die Datenbank geschrieben. Genau wie bei den Transaktionen gibt es auch für Blöcke gewisse Konsensregeln. Sobald ein Block allen Konsensregeln entspricht, ist er bereit in die Datenbank aufgenommen zu werden. Da sich alle Netzwerkteilnehmer über die Gültigkeit des Blocks einig sind, wird somit die globale Blockchain Datenbank angepasst. Genau wie bei den Transaktionen ist auch die Reihenfolge der Blöcke wichtig. Daher beinhaltet jeder Block den Hash-Wert seines Vorgängers (siehe Abbildung 2.3).

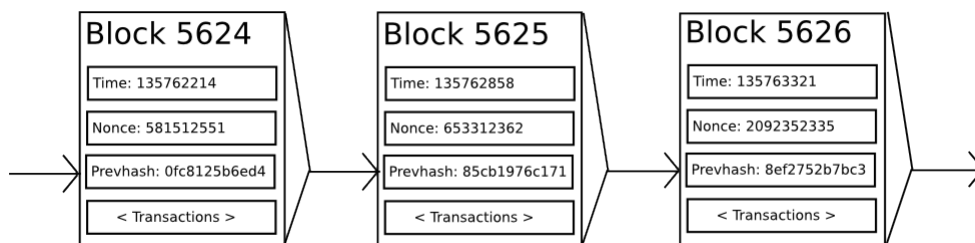


ABBILDUNG 2.3: Verkettung von Blöcken [6]

Durch die so erzielte Verkettung der Blöcke wird die Reihenfolge eindeutig festgelegt und es entsteht die sogenannte Blockchain. Am Anfang der Blockchain befindet sich der sogenannte Genesis Block. Auf diesen bauen alle weiteren Blöcke auf. Nachträgliche Änderungen an einem bereits eingefügten Block sind nicht möglich, da sich dadurch der Blockhash des Blocks verändert und somit die Kette an dieser Stelle "zerbricht".

2.1.3 Konsensregeln

Konsensregeln sind Regeln, über die sich alle Teilnehmer des Peer-to-Peer-Netzwerks einig sind. Sie stellen sicher, dass die grundlegenden Eigenschaften der Kryptowährung eingehalten werden. Bei Bitcoin gibt es eine ganze Reihe an Konsensregeln. Die wichtigsten sind:

- Transaktionen dürfen kein Geld aus dem nichts schöpfen, sondern nur bereits existierende Beträge von einer Adresse auf eine andere Adresse überweisen. Die Blockreward-Konsensregel bildet hierzu die einzige Ausnahme. Sie legt fest wie neue Währungseinheiten erschaffen werden.
- Blockreward: Ein neuer Block muss genau eine Transaktion enthalten, die neue Kryptowährungseinheiten aus dem Nichts erschafft. Sowohl die Höhe des Betrags als auch die Anpassung des Betrags über die Zeit ist in den Konsensregeln hart verankert. Bei Bitcoin halbiert sich der Blockreward jedes Mal nach einer festgelegten Zeitspanne. Dies stellt sicher, dass es eine feste Obergrenze an Währungseinheiten gibt.
- Transaktionen, die Geld von Adresse A nach Adresse B überweisen, müssen durch eine Signatur beweisen, dass sie von dem rechtmäßigen Besitzer stammen.
- Blockgröße: Diese Konsensregel legt die maximale Größe eines Blocks in Megabyte fest. Sie beeinflusst wie viele Transaktionen in einem Block gebündelt werden können. Dies ist wichtig, da sie zusammen mit der Blockzeit-Konsensregel das Wachstum der Blockchain-Datenbank steuert.
- Blockzeit: Diese legt fest in welchem durchschnittlichen Zeitabstand es erlaubt ist einen neuen Block in die Blockchain einzufügen. Bei Bitcoin ist diese Zeit auf 10 Minuten festgelegt.
- Längste Blockchain-Kette: Teilnehmer des Peer-to-Peer Netzwerks folgen immer der Kette, die am meisten Proof-of-Work beinhaltet und betrachten "kürzere" Ketten als ungültig.

Die Konsensregeln ermöglichen es, dass jeder Teilnehmer eigenständig lokal seine Version der Blockchain Datenbank verwalten kann, ohne dabei einem anderen Teilnehmer vertrauen zu müssen. Die Konsensregeln stellen somit sicher, dass alle Teilnehmer die gleiche Blockchain Datenbank lokal aufbauen und sich dadurch auf den Systemzustand einigen.

2.1.4 Proof-of-Work und Mining

Bei einer dezentralen Währung gibt es keine zentrale Instanz, die eine Anpassung des Systemzustands koordiniert, beziehungsweise autorisiert. Um dieses Problem zu lösen, verwendet Bitcoin einen Proof-of-Work Algorithmus. Dieser basiert auf der Idee, dass Teilnehmer des Netzwerks nachweisen müssen, dass sie einen gewissen Aufwand betrieben haben, bevor sie einen neuen Block an die Blockchain anhängen dürfen. Um den nächsten gültigen Block zu produzieren, muss der Hashwert des Blocks unter einem dynamisch angepassten Schwierigkeit-Zielwert liegen. Der Schwierigkeit-Zielwert wird durch die Blockzeit Konsensregel so angepasst, dass im gesamten Netzwerk im Durchschnitt alle 10 Minuten ein neuer Bitcoin Block gefunden wird.

Bei der Suche nach einem gültigen Block wird nach jeder Berechnung des Hashwerts das nonce-Feld des Blocks erhöht, damit ein neuer Hashwert entsteht. Findet ein Teilnehmer den Blockhash eines den Konsensregeln entsprechenden Blocks, leitet er diesen Block an das Peer-to-Peer Netzwerk weiter und erhält im Gegenzug den Blockreward. Teilnehmer, die unbestätigte¹ Transaktionen empfangen, diese in Blöcken zusammenfassen und unter Zuhilfenahme des Proof-of-Work Algorithmus in die Blockchain einfügen, werden "Miner" genannt. Dieser Name stammt daher, dass bei diesem rechenintensiven Prozess gleichzeitig neue Bitcoins erschaffen werden.

Beim Mining werden die in Tabelle 2.1 aufgeführten Felder des Blockheaders als Eingabe für die kryptographische Hashfunktion SHA256 verwendet².

TABELLE 2.1: Bitcoin Blockheader

Feld	Verwendungszweck	Aktualisiert falls...	Bytes
version	Block Versionsnummer	man die Software aktualisiert und diese eine neue Version spezifiziert.	4
hashPrevBlock	256-bit Hash des vorherigen Blockheaders	ein neuer gültiger Block empfangen wird.	32
hashMerkleRoot	256-bit Hash aller Transaktionen des Blocks	eine neue Transaktion akzeptiert wird.	32
time	Aktueller Zeitstempel in Sekunden seit 1970-01-01	Alle paar Sekunden...	4
bits	Aktuelles Schwierigkeitsziel	wenn das Schwierigkeitsziel angepasst wird.	4
nonce	32-bit Nummer (von 0 aus erhöht)	der Hash ausprobiert wurde. (nonce+1)	4

Die Transaktionen des Blocks gehen nicht direkt, sondern nur durch den sogenannten Merkel Tree Hash³ in den Blockhash mit ein. Das nonce-Feld des Blockheaders wird nach jedem Hashversuch um den Wert 1 erhöht. Dadurch wird die Ausgabe der Hashfunktion kontinuierlich verändert.

Die Sicherheit des Bitcoin Netzwerkes und die Unmanipulierbarkeit der Blockchain ergeben sich aus der im Protokoll verankerten Spieltheorie. Die Spieltheorie macht es für einen Miner profitabler sich an die Spielregeln des Protokolls zu halten als zu versuchen das Netzwerk zu betrügen. Versucht ein Miner einen Block zu produzieren, der den Konsensregeln widerspricht, wird dieser vom Netzwerk verworfen. Somit hält kein anderer Netzwerkteilnehmer den vom Miner an sich selbst ausgeschütteten Blockreward für gültig. Da der Miner aber Ausgaben in Form von Hardwareabnutzung und Stromkosten zu bezahlen hat, schafft dies einen Anreiz sich den Konsensregeln zu unterwerfen.

¹ Alle gültigen Transaktionen, die noch nicht Teil der längsten Blockchain sind, gelten als unbestätigt.

² Um genau zu sein wird der Blockheader bei Bitcoin zweimal mit der SHA 256 Hashfunktion gehasht. Blockhash = SHA256(SHA256(Blockheader))

³ Der genaue Aufbau des Merkel Trees und dessen Vorteile sind in [28] genauer beschrieben.

2.2 Konzept

Die folgenden Schritte beschreiben den Finanzfluss zwischen den Teilnehmern und der Anwendung, sowie die Gewinnerauswahl durch den in der Zukunft liegenden Blockchain-Status. Der Ablauf ist allgemein gehalten und kann nicht nur mit Bitcoin, sondern auch mit anderen Kryptowährungen, die auf einer Proof-of-Work Blockchain basieren, umgesetzt werden. Betrachtet wird ein Spiel mit $N \in \{2, 5, 10\}$ Teilnehmern, bei dem jeder Teilnehmer einen Einsatz von X Währungseinheiten zur Teilnahme zahlen muss.

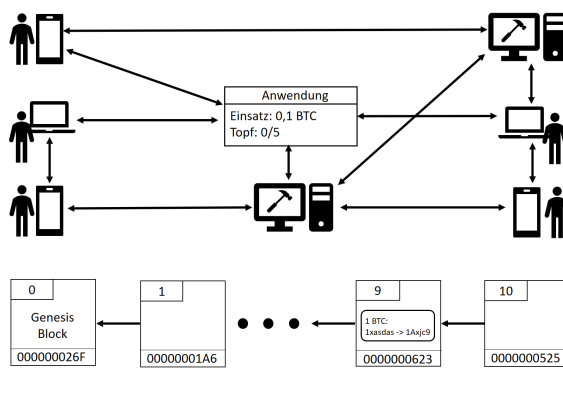
1. Im ersten Schritt eröffnet die Anwendung ein neues Spiel in dem es N freie Plätze und einen leeren Geldtopf gibt.
2. Sobald ein Spieler am Spiel teilnehmen möchte, generiert die Anwendung eine neue Empfangsadresse⁴ und zeigt diese dem Spieler an.
3. Der Spieler verwendet die Wallet Software seiner Wahl um eine Transaktion zu erstellen, die den Einsatz an die angezeigte Empfangsadresse überweist. Die Wallet Software signiert die Transaktion und leitet sie über die mit ihr verbundenen Nachbarknoten an das Peer-to-Peer Netzwerk weiter.
4. Sowohl die Anwendung als auch die Miner empfangen die Transaktion. Die Anwendung zeigt dem Nutzer über die GUI an, dass die Transaktion zwar erhalten wurde, allerdings noch nicht bestätigt wurde. Die Miner des Netzwerks nehmen die Transaktion in ihren nächsten Block auf.⁵
5. Ein Miner findet den zu seinem Block passenden Proof-of-Work-Hash und schickt den Block an das Netzwerk.
6. Die Applikation empfängt den Block und merkt, dass im Block eine Transaktion auf die in Schritt 2 generierte Empfangsadresse enthalten ist. Die Applikation prüft die Höhe des Transaktionsbetrags und leitet anschließend die vom Spieler kontrollierte Auszahlungsadresse aus der Transaktion ab.
7. Die restlichen $N - 1$ Teilnehmer überweisen ebenfalls den geforderten Betrag auf die ihnen angezeigte Empfangsadresse.
8. Sobald die letzte Transaktion in einen validen Block aufgenommen wurde, zählt die Reihenfolge in der die Transaktionen in der Blockchain stehen. Die Reihenfolge steht somit fest und kann nicht mehr nachträglich verändert werden. Der Geldtopf ist nun mit einem Betrag von $N * X$ Kryptowährungseinheiten gefüllt und wird geschlossen. Der Nachfolger des Blocks, in dem die letzte Einzahlungstransaktion eingegangen ist, wird zur Gewinnerauswahl genutzt. Die Anwendung merkt sich die Blocknummer dieses Blocks.
9. Die Anwendung und die Teilnehmer warten darauf, dass der nächste Block von einem Miner gefunden wird. Alle Miner des Peer-to-Peer Netzwerks versuchen schnellstmöglich einen passenden Blockhash zu finden, um den Blockreward zu erhalten. Ein Miner gewinnt dieses Rennen und teilt dem Netzwerk den neu gefundenen Block mit.

⁴Man könnte auch allen Spielern die gleiche Einzahlungsadresse anzeigen. (siehe Abschnitt 2.4.5)

⁵Natürlich nur unter der Annahme, dass sie die Höhe der Transaktionsgebühr als angemessen empfinden.

10. Die Anwendung empfängt den nächsten Block und ermittelt durch diesen den Gewinner. Die Berechnung erfolgt auf Basis der letzten Ziffer⁶ d_{last} des Blockhashs in Hexadezimaldarstellung. Durch die Berechnung von d_{last} modulo N resultiert eine Zahl $G \in \{0, \dots, N - 1\}$, die den Gewinner festlegt. Der Spieler, der die $G + 1$ te Einzahltransaktion gesendet hat, gewinnt den Geldtopf⁷.
11. Sobald die Anwendung einen Block empfängt, der auf den Block für die Gewinnerauswahl aufbaut, beginnt die Auszahlung des Gewinns. Die Anwendung erstellt dazu eine neue Transaktion, die alle $N * X$ Kryptowährungseinheiten des Geldtopfs an die Auszahlungsadresse des Gewinners überweist und sendet diese an das Netzwerk.
12. Die Wallet Software des Gewinners, empfängt die Transaktion und informiert ihn darüber, dass er den gesamten Betrag des Topfes erhalten hat.

Im folgenden Beispiel wird ein Topf mit 5 Teilnehmern, die Kryptowährung Bitcoin und einen Einzahlungsbetrag von 0,1 Bitcoin betrachtet. Dieses Beispiel verdeutlicht sowohl die Interaktion der verschiedenen Teilnehmer des Peer-to-Peer Netzwerks, als auch die Veränderung des Status der Blockchain.



Diese Abbildung zeigt das Peer-to-Peer Netzwerk. Die 5 potentiellen Teilnehmer sind durch Notebooks und Smartphones dargestellt. Außerdem sind 2 Miner und die Glücksspielanwendung Teil des Peer-to-Peer Netzwerks. Der aktuelle Status der Blockchain, die jeder Teilnehmer des Netzwerks lokal speichert, ist unterhalb des Netzwerks dargestellt.

ABBILDUNG 2.4: Schritt 1

⁶Man kann an dieser Stelle auch den gesamten Blockhash als Grundlage der Gewinnerauswahl nehmen. Die daraus resultierenden Vor- und Nachteile werden in Abschnitt 2.4.2 erörtert.

⁷Es handelt sich zu diesem Zeitpunkt erst um den vorläufigen Gewinner des Spiels, da es zu einem Blockchain Fork (siehe Abschnitt 2.4.7) kommen kann.

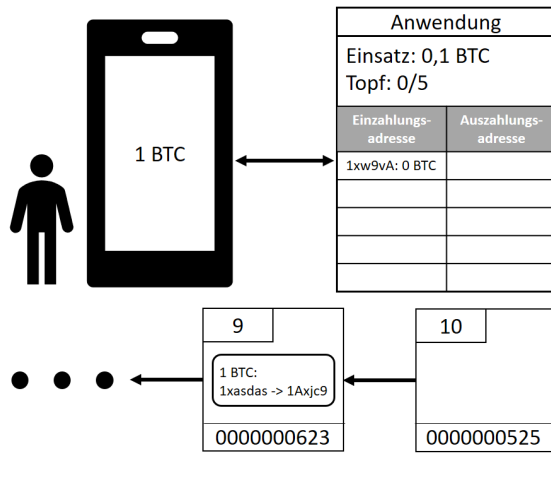


ABBILDUNG 2.5: Schritt 2

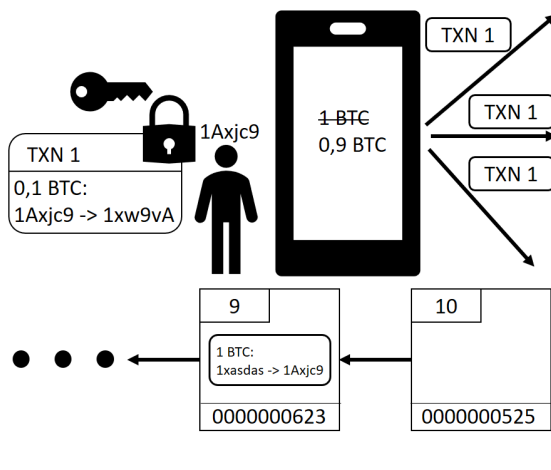


ABBILDUNG 2.6: Schritt 3

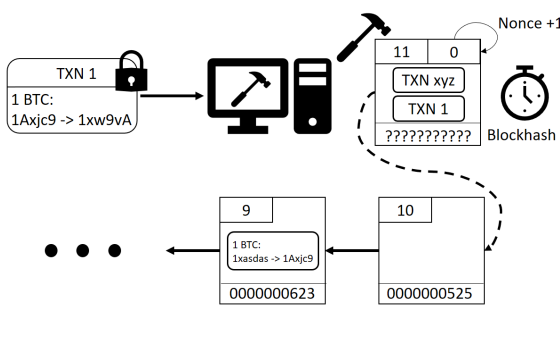


ABBILDUNG 2.7: Schritt 4

Die Bitcoin Client Software der Glücksspielanwendung generiert eine neue Bitcoinadresse und speichert den dazugehörigen privaten Schlüssel in der Wallet. Sobald Bitcoins auf dieser Adresse empfangen werden, können sie nur durch den Besitz des privaten Schlüssels weiter transferiert werden. Die Anwendung zeigt dem Benutzer eine frisch generierte Empfangsadresse über die Benutzeroberfläche an. Der Zustand der Blockchain verändert sich nicht.

Nun zahlt der Spieler mithilfe seiner Bitcoin Wallet Software in den Geldtopf ein. Dazu erstellt er eine Transaktion (TXN 1), die Bitcoin von seiner Adresse auf die generierte Adresse der Glücksspielanwendung transferiert. Durch die Signierung mit seinem privaten Schlüssel autorisiert er die Überweisung. Anschließend schickt er die Transaktion seinen Nachbarknoten.

Sobald die Transaktion TXN 1 einen Miner erreicht, prüft dieser, ob die Transaktion in Einklang mit den Konsensregeln ist. In diesem Beispiel existiert in Block 9 eine Transaktion von einem Bitcoin auf die Adresse des Teilnehmers. Unter der Annahme, dass dieser Bitcoin nicht in Block 10 weiter überwiesen wurde, befindet sich auf der Adresse des Teilnehmers somit ein Bitcoin. Außerdem prüft der Miner ob die Signatur der Transaktion gültig ist. Da die Transaktion valide ist, fügt er sie dem aktuell zu generierenden Block Nummer 11 hinzu.

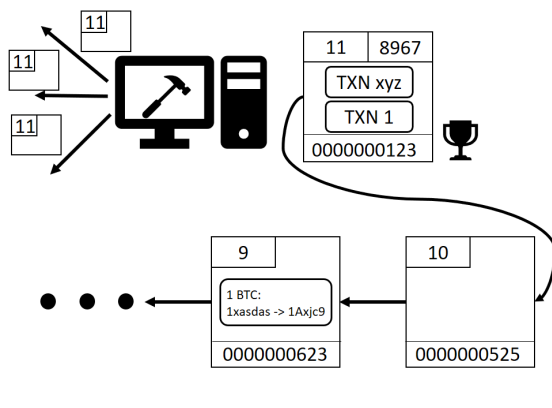


ABBILDUNG 2.8: Schritt 5

Der Miner berechnet nun mithilfe der Hashfunktion den Hash des Blocks. Falls der Blockhash-Wert den durch die Konsensregeln dynamischen angepassten Schwierigkeits-Wert unterschreitet, gilt der Block als valide. Überschreitet der Blockhash den Wert erhöht der Miner den Nonce-Wert des Blocks und berechnet den Blockhash erneut. Diesen Prozess wiederholt er solange bis er entweder einen gültigen Blockhash findet oder einen gültigen Block Nummer 11 von einem anderen Netzwerkteilnehmer empfängt. In diesem Beispiel findet der Miner einen gültigen Blockhash, leitet den Block an das Netzwerk weiter und wird dadurch mit neu erschaffenen Bitcoin für seinen Rechenaufwand belohnt.

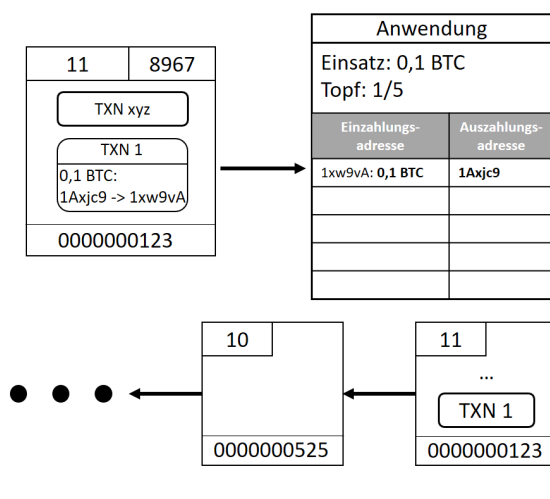


ABBILDUNG 2.9: Schritt 6

Die Glücksspielanwendung empfängt den Block Nummer 11 und überprüft, ob er im Einklang mit den Konsensregeln ist. Dies ist der Fall. Somit wird die lokale Blockchain Datenbank um einen Block erweitert. Die Glücksspielanwendung hat somit den Einsatz des ersten Spielers erhalten. Sie zeigt nun die vorher noch unbestätigte Transaktion als bestätigt an und fügt den Spieler zum Topf hinzu. Aus der Einzahlungstransaktion des Spielers leitet die Anwendung die Auszahlungsadresse **1Axjc9** des Spielers ab.

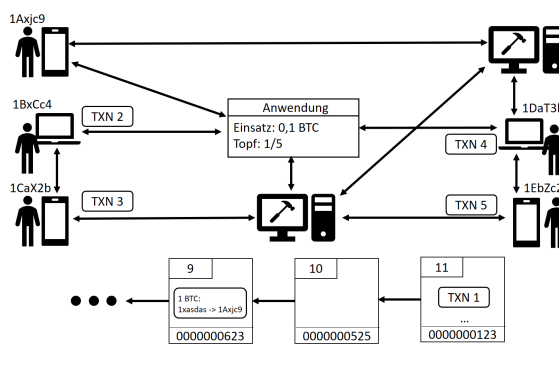


ABBILDUNG 2.10: Schritt 7

Die restlichen Spieler senden ihre signierten 0,1 Bitcoin Transaktionen in das Peer-to-Peer Netzwerk. Diese sind in Abbildung 7 durch die Transaktionen TXN 2 bis 5 dargestellt.

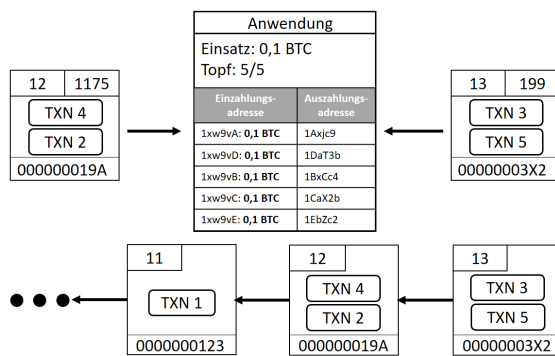


ABBILDUNG 2.11: Schritt 8

Beliebige Miner fügen die Transaktionen in ihre Blöcke ein. Sobald die Glücksspielanwendung die Blöcke empfängt, prüft sie diese gegen die Konsensregeln und fügt sie in die lokale Blockchain ein. Die Applikation merkt nun, dass alle Spieler bezahlt haben und schließt den Geldtopf. Dabei merkt sie sich die Nummer des Blocks, in dem die letzte Einzahlungstransaktion vorhanden ist. Der darauffolgende Block mit Nummer 14 wird für die Ziehung des Gewinners verwendet.

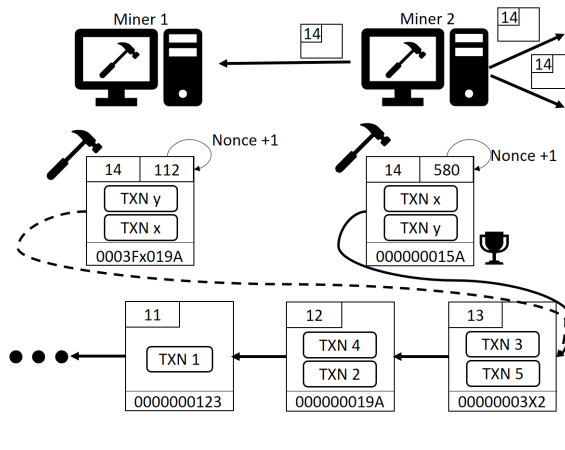


ABBILDUNG 2.12: Schritt 9

Alle Miner des Netzwerks versuchen nun gleichzeitig so schnell wie möglich den nächsten Block zu finden. Da sie dazu eine kryptographische Hashfunktion benutzen, bei der die Ausgabe ein unkontrollierbarer, zufälliger Wert ist, hat keiner der Miner einen direkten Einfluss auf den resultierenden Blockhash. In diesem Beispiel findet Miner 2 einen gültigen Blockhash vor Miner 1. Miner 2 leitet seinen gültigen Block Nummer 14 so schnell wie möglich an das Netzwerk weiter und erhält den Blockreward als Belohnung. Miner 1 geht leer aus.

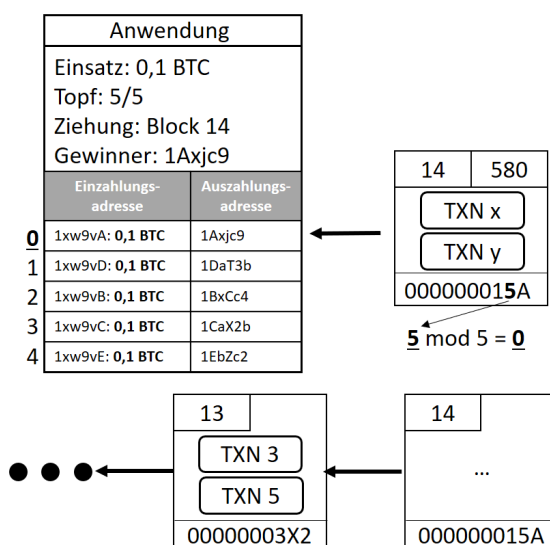


ABBILDUNG 2.13: Schritt 10

Die Anwendung empfängt Block 14 und prüft ihn gegen die Konsensregeln. Der Block ist valide. Daher verwendet die Anwendung den im Block enthaltenen Blockhash um den Gewinner des Geldtopfs zu ermitteln. Statt des gesamten Blockhashs verwendet die Anwendung nur die letzte Ziffer des Blockhashs in Hexadezimaldarstellung zur Gewinnerauswahl. Dies hat den Vorteil, dass die Teilnehmer die Korrektheit der Gewinnerauswahl leichter eigenständig nachprüfen können. Da die letzte numerische Stelle des Blockhashs 10 verschiedene Werte annehmen kann, ordnet die Anwendung jedem der 5 Teilnehmer zwei Gewinnzahlen zu.

Durch die Modulo 5 Funktion ergibt sich die Verteilung der Gewinnzahlen folgendermaßen:

- Spieler 1 mit Adresse 1xw9vA gewinnt bei 0 und 5,
- Spieler 2 mit Adresse 1xw9vD gewinnt bei 1 und 6,
- Spieler 3 mit Adresse 1xw9vB gewinnt bei 2 und 7,
- Spieler 4 mit Adresse 1xw9vC gewinnt bei 3 und 8,
- Spieler 5 mit Adresse 1xw9vE gewinnt bei 4 und 9.

Jeder Teilnehmer besitzt nun die gleiche Gewinnwahrscheinlichkeit⁸ von $\frac{1}{5}$. Block Nummer 14 hat den Blockhash **000000015A**⁹. Die zur Gewinnerauswahl benutzte Ziffer ist somit die 5. Da 5 modulo 5 den Wert 0 ergibt, gewinnt Spieler 1 mit der Adresse **1xw9vA**.

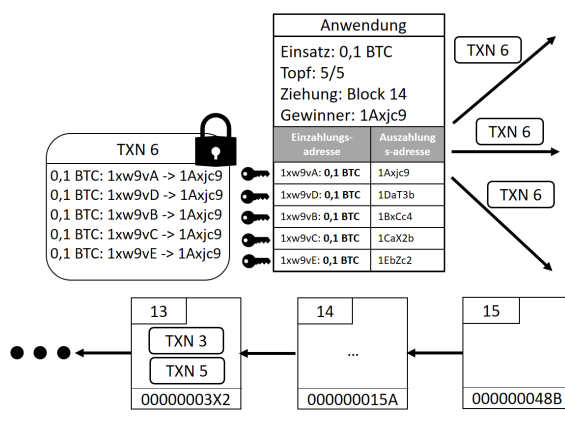


ABBILDUNG 2.14: Schritt 11

Die Anwendung erstellt nun eine Transaktion, die alle Spieleinsätze an die Auszahlungsadresse **1Axc9** von Spieler 1 überweist. Um die Transaktion zu signieren, verwendet die Anwendung die zu den 5 Einzahlungsadressen passenden privaten Schlüssel. Anschließend leitet die Anwendung die Transaktion an das Peer-to-Peer Netzwerk weiter.

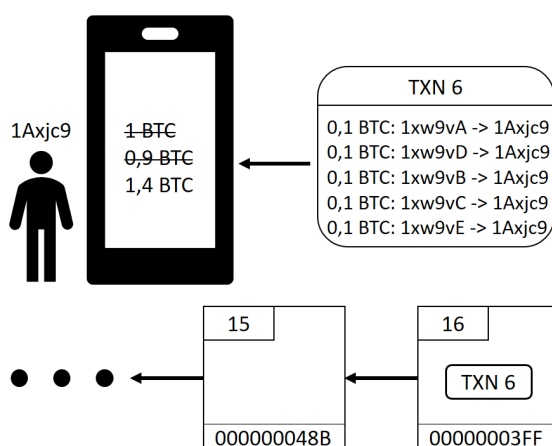


ABBILDUNG 2.15: Schritt 12

Das Smartphone von Spieler 1 empfängt die Transaktion gegebenenfalls noch bevor sie von einem Miner in einen validen Block aufgenommen wurde. Die Wallet Software zeigt die Transaktion erst als unbestätigt an. Sobald sie durch die Aufnahme in Block 16 bestätigt wurde, gilt sie für die Wallet Software als bestätigt.

⁸Dies gilt nur unter der Annahme, dass der Wert für die Gewinnerauswahl gleichverteilt ist. Dies wird in Abschnitt 2.4.3 genauer erläutert.

⁹In der Praxis ist jeder Blockhash genau 64 Zeichen lang (Hexadezimalsystem), da Bitcoin die SHA-256 Hashfunktion verwendet. Aufgrund der Länge ist es praktisch unmöglich, dass der Blockhash ausschließlich aus Buchstaben besteht.

Abbildung 2.16 zeigt außerdem:

- Peer Discovery, Peer Datenbank und Connection Manager: Diese kümmern sich um die Kommunikation mit dem Peer-to-Peer Netzwerk.
- Mempool: Im sogenannten Mempool werden empfangene, unbestätigte Transaktionen im Speicher gehalten.
- Validation Engine: Diese validiert, ob die empfangenen Blöcke und deren Transaktionen die Konsensregeln einhalten. Falls ja, werden die somit bestätigten Transaktionen aus dem Mempool gelöscht und der Block wird an die Storage-Engine zur Abspeicherung in der Blockchain Datenbank weitergereicht.
- Miner: Die Bitcoin Full-Node Software enthält einen CPU Miner mit dem man nach neuen Blöcken suchen kann. Bitcoin Mining ist heutzutage allerdings nur noch mit sogenannten ASICs profitabel. ASIC steht für **A**pplication-**S**pecific **I**ntegrated **C**ircuit. Es handelt sich um Hardware, die auf eine möglichst performante Berechnung der SHA256 Hashfunktion spezialisiert ist.

Light-Node

Light-Nodes speichern nicht die gesamte Blockchain sondern in der Regel nur die Blockheader der Blöcke der Blockchain. Beim Mining gehen nur die Daten des Blockheaders in den Blockhash ein (siehe Tabelle 2.1). Der Node empfängt Blockheader, prüft ihre Gültigkeit und fügt sie gegebenenfalls in die Headerkette ein. Der Node kann somit eigenständig, d.h. ohne seinen Nachbarn vertrauen zu müssen, die längste Proof-of-Work Kette bilden. Da diese Kette nur aus Headern besteht und keine Transaktionen enthält, kann der Light-Node empfangene Transaktionen nicht eigenständig auf ihre Gültigkeit prüfen. Light-Nodes verwenden das in [28] beschriebene **S**implified **P**ayment **V**erification Verfahren zur Prüfung von Transaktionen. Light-Nodes werden daher oft auch *SPV-Clients* genannt. Ein **SPV-Client** prüft die Gültigkeit einer Transaktion, indem er sie an der richtigen Stelle der Headerkette einordnet und dann den passenden Merkle-Branch von einem Full-Node anfragt. Durch diese zusätzlichen Daten kann er nun, wie in Abbildung 2.17 gezeigt, nachprüfen, ob der Hash der Transaktion wirklich in den Wurzelknoten des Merkle-Trees mit eingegangen ist.

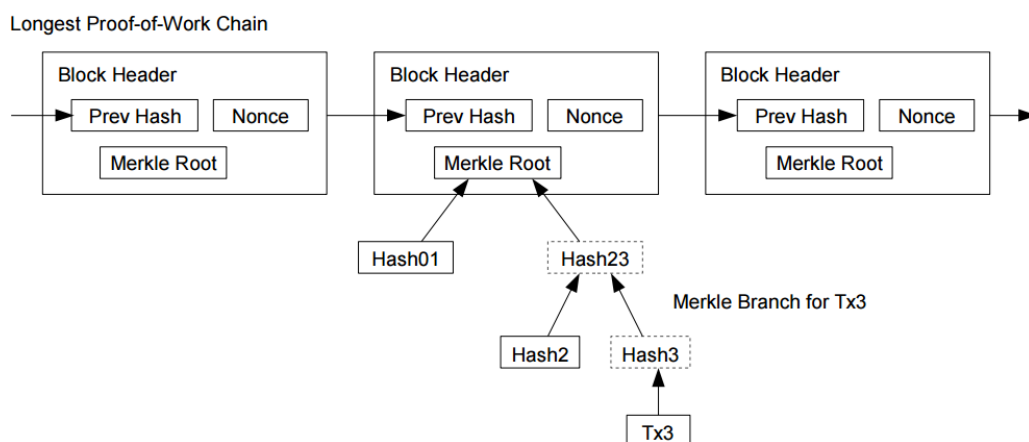


ABBILDUNG 2.17: Blockheader Kette [6]

Für den Bitcoin-Teil dieser Ausarbeitung ist die Integration mit dem Peer-to-Peer Netzwerk mit Hilfe der in Java geschriebenen BitcoinJ [4] Bibliothek umgesetzt. Diese ermöglicht die Interaktion mit dem Netzwerk als SPV-Client.

2.3.2 Überblick

Abbildung 2.18 skizziert die Komponenten der Glücksspielanwendung und wie diese mit ihrer Umgebung kommunizieren. Es gibt den Server, auf dem die Glücksspielanwendung läuft, die Spieler und das Bitcoin Netzwerk.

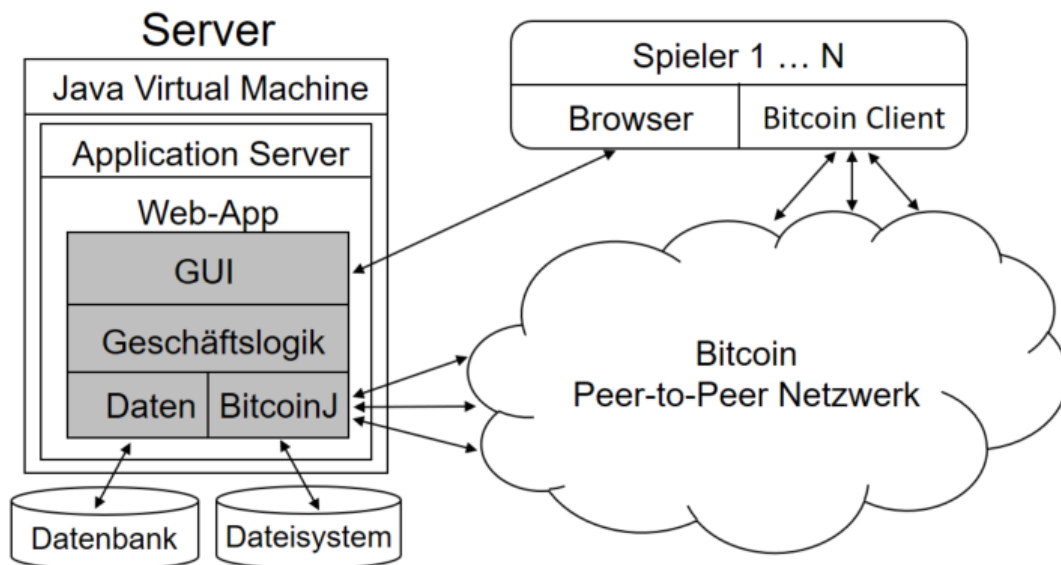


ABBILDUNG 2.18: Glücksspielanwendung Aufbau und Interaktion

Glücksspielanwendung

- **Server:** Die Glücksspielanwendung läuft auf einem Server, der eine Java Virtual Machine Laufzeitumgebung und ein gewöhnliches Dateisystem besitzt. Außerdem ist eine Datenbank zur Abspeicherung der Spieldaten notwendig.
- **Java Virtual Machine (JVM):** Innerhalb der JVM läuft ein sogenannter Application Server, der eine Webanwendung nach außen bereitstellt. Auf diese Webanwendung können die Spieler über das HTTP Protokoll mittels ihres Browsers zugreifen. Die Webanwendung besteht aus mehreren Komponenten.
- **Application Server:** Dieser stellt die Applikation bereit. Bei der Umsetzung der Glücksspielanwendung wurde der Open Source Application Server Wildfly¹⁰ von der Firma Red Hat verwendet.
- **GUI:** Die Weboberfläche stellt die zentrale Schnittstelle zwischen der Anwendung und dem Spieler dar. Diese ist mithilfe des Tapestry¹¹ Webframeworks von Apache umgesetzt. Detaillierte Informationen findet man in [12].
- **Geschäftslogik:** Diese behandelt die vom Benutzer und vom Bitcoin Netzwerk ausgelösten Events.

¹⁰<http://wildfly.org/>

¹¹<http://tapestry.apache.org/>

- BitcoinJ: Die Java Bibliothek, die zur Kommunikation mit dem Bitcoin Netzwerk verwendet wird.

Spieler

Die Spieler verfügen über einen Internetbrowser und über einen Bitcoin Client. Mit dem Browser interagieren sie mit der Glücksspielanwendung. Mit dem Bitcoin Client initiieren und empfangen sie Zahlungen.

Bitcoin Peer-to-Peer Netzwerk

Das Peer-to-Peer Netzwerk besteht aus Full-Nodes, Light-Nodes und Miner. Bei Kryptowährungsnetzwerken unterscheidet man in der Regel zwischen dem Test- und Hauptnetzwerk. Den Bitcoins des Testnetzwerks wird kein monetärer Wert zugeschrieben. Das Testnetzwerk ermöglicht es Software, die mit dem Bitcoin Netzwerk interagieren soll, zu testen. Möchte beispielsweise ein Händler Bitcoin in seinen Webshop integrieren, kann er so seine Implementierung testen, ohne ein finanzielles Risiko einzugehen.

2.3.3 Datenmodell

Die Klasse Pot repräsentiert ein Spiel und speichert alle für das Spiel relevanten Daten. Sie besteht aus einer Liste von Teilnehmern (Participant). Jeder Teilnehmer besitzt, wie im Konzept beschrieben, eine Ein- und Auszahlungsadresse.

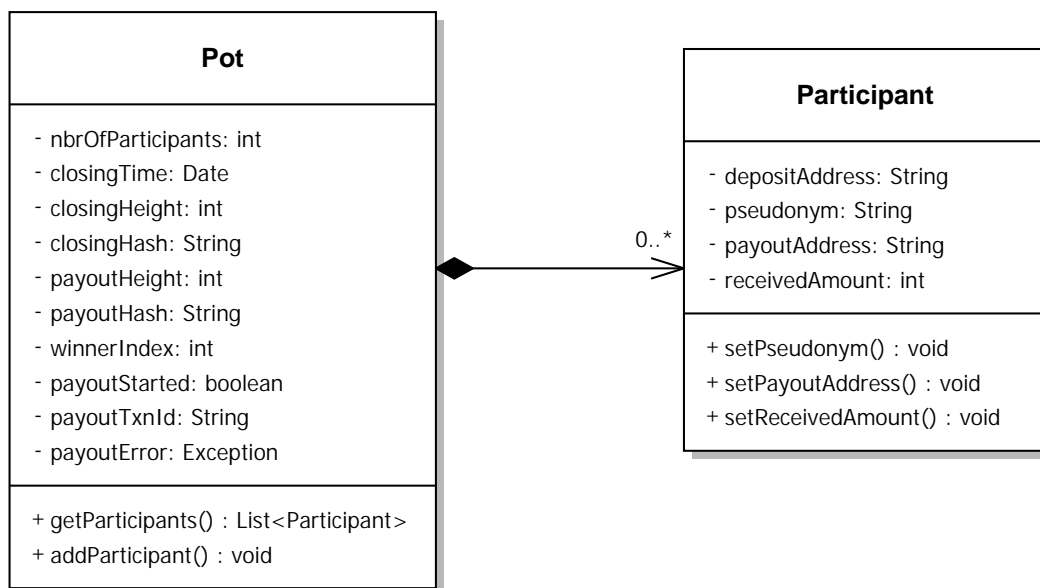


ABBILDUNG 2.19: Datenmodell Klassendiagramm

2.3.4 Geschäftslogik

Die Java Klassen der Glücksspielanwendung können, wie in Abbildung 2.20 gezeigt, in 3 verschiedene Gruppen unterteilt werden. Das **Core Module** enthält die Klassen des Datenmodells und ein Interface mit dem die GUI Anwendung interagiert.

Das Interface entkoppelt die Anzeigelogik des GUI-Quellcodes von der Geschäftslogik der Kryptowährung. Die GUI Komponente bekommt von der Schnittstelle allgemeine Daten und kümmert sich nur um deren Anzeige. Das **Bitcoin Service Module** enthält die gesamte kryptowährungsspezifische Geschäftslogik. **BitcoinJ** enthält alle Klassen und Interfaces, die benötigt werden um mit dem Bitcoin Netzwerk zu interagieren und Daten aus der Blockchain auszulesen.

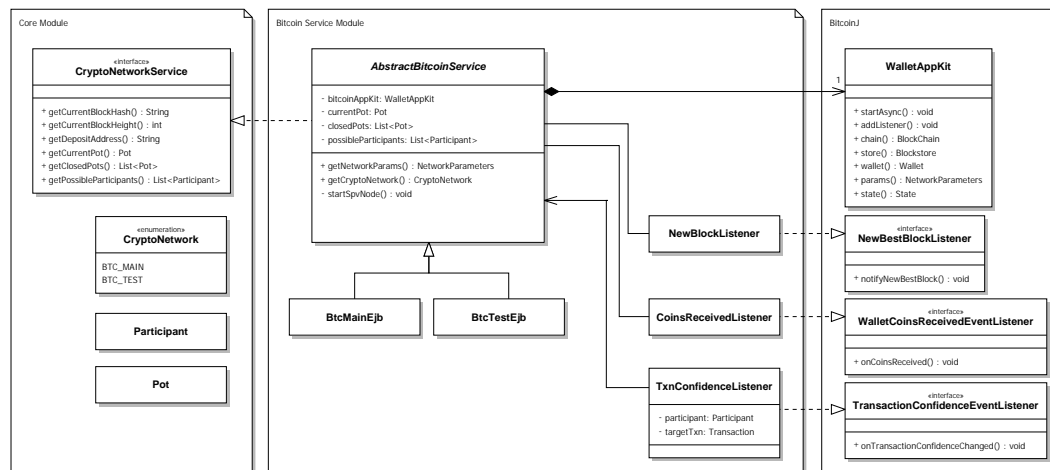


ABBILDUNG 2.20: Geschäftslogik Klassendiagramm

Start der Anwendung

Beim Kompilieren der Anwendung wird über einen Konfigurationseintrag festgelegt, ob die Anwendung mit dem Bitcoin Haupt- oder Testnetzwerk interagieren soll. Für das Hauptnetzwerk wird die Java Klasse `BtcMainEjb` verwendet. Für das Testnetzwerk wird die Klasse `BtcTestEjb` verwendet. Beide Klassen verwenden die gleiche Implementierung der abstrakten Oberklasse `AbstractBitcoinService`. Diese wiederum implementiert das von der GUI Komponente verwendete `CryptoNetworkService` Interface.

```

1 import javax.ejb.Startup;
2 import org.bitcoinj.params.AbstractBitcoinNetParams;
3 import org.bitcoinj.params.MainNetParams;
4 import com.ossel.gamble.bitcoin.services.AbstractBitcoinService;
5 import com.ossel.gamble.core.data.enums.CryptoNetwork;
6
7 @Startup
8 @Singleton
9 public class BtcMainEjb extends AbstractBitcoinService {
10     @Override
11     public CryptoNetwork getCryptoNetwork() {
12         return CryptoNetwork.BTC_MAIN;
13     }
14     @Override
15     public AbstractBitcoinNetParams getNetworkParams() {
16         return MainNetParams.get();
17     }
18 }

```

Die beiden Klassen `BtcMainEjb` und `BtcTestEjb` sind mit den Annotationen `@Startup` und `@Singleton` markiert. Es handelt sich um sogenannte **Enterprise Java Beans**. Dies bedeutet, dass der Applikationsserver diese eigenständig verwaltet und genau eine Instanz der Klasse beim Start der Applikation erzeugt. Beim Start wird dann die mit `@PostConstruct` annotierte `startSpvNode` Methode aufgerufen und abgearbeitet. Diese konfiguriert und startet das `WalletAppKit`, welches die zentrale Klasse zur Interaktion mit der BitcoinJ Bibliothek darstellt.

```

1  @PostConstruct
2  private void startSpvNode() {
3      log.info("#### Start Bitcoin SPV Node ####");
4      currentPot = new Pot(2, 100000L);
5      File walletDir = CoreUtil.getWalletDirectory();
6      NetworkParameters params = getNetworkParams();
7      String fileName = "bitcoin-" + params.getPaymentProtocolId();
8      bitcoinAppKit = new WalletAppKit(params, walletDir, fileName) {
9          @Override
10         protected void onSetupCompleted() {
11             log.info("#### Bitcoin SPV Node started ####");
12         }
13     };
14     bitcoinAppKit.startAsync();
15     waitUntilStarted(bitcoinAppKit);
16     newBlockListener = new NewBlockListener(this);
17     bitcoinAppKit.chain().addNewBestBlockListener(newBlockListener);
18     coinReceivedListener = new CoinsReceivedListener(this);
19     bitcoinAppKit.wallet().addCoinsReceivedEventListener(coinReceivedListener);
20 }

```

In Zeile 4 wird zunächst ein neuer, leerer Topf mit 2 Teilnehmern erzeugt. Anschließend wird das `WalletAppKit` erzeugt. Dazu bekommt dieses die gewünschten Netzwerkparameter und den Pfad zum Dateisystem übergeben. Unter diesem Dateipfad speichert BitcoinJ die Blockchain und Wallet-Daten. Zeile 14 startet den durch das `WalletAppKit` repräsentierten SPV-Node. Anschließend wird dem `WalletAppKit` noch ein `NewBlockListener` und ein `CoinsReceivedListener` hinzugefügt, um auf neue Blöcke und eingehende Zahlungen zu reagieren.

GUI Events

Der folgende Code zeigt die Implementierung der Methoden, die von der GUI Komponente aufgerufen werden.

```

1  public Pot getCurrentPot() {
2      return currentPot;
3  }
4
5  public String getDepositAddress() {
6      String depositAddress =
7          bitcoinAppKit.wallet().freshAddress(KeyPurpose.RECEIVE_FUNDS).toString();
8      possibleParticipants.add(new Participant(depositAddress));
9      return depositAddress;
10 }
11
12 public String getCurrentBlockHash() {
13     return
14         bitcoinAppKit.chain().getChainHead().getHeader().getHash().toString();
15 }

```

```

15
16 public int getCurrentBlockHeight() {
17     return bitcoinAppKit.chain().getChainHead().getHeight();
18 }

```

Die Methode `getCurrentPot` reicht in Zeile 2 die Daten des Topfs an die GUI Komponente weiter. Zeile 6 erzeugt eine neue Empfangsadresse und fügt einen weiteren möglichen Teilnehmer mit dieser Adresse der `possibleParticipants` Liste hinzu. Erst wenn eine Zahlung auf diese Adresse eingeht, wird der Teilnehmer dem Topf hinzugefügt. Zeile 13 gibt den Blockhash des neusten Blocks der Blockheader-Kette zurück. Zeile 17 gibt die Blocknummer des neusten Blocks der Blockheader-Kette zurück.

Bitcoin Netzwerk Events

Immer wenn der SPV-Node eine Transaktion auf eine vorher erzeugte Adresse¹² empfängt, wird die `onCoinsReceived` Methode der Klasse `CoinsReceivedListener` aufgerufen.

```

1 @Override
2 public void onCoinsReceived(Wallet wallet, Transaction txn, Coin
    prevBalance, Coin newBalance) {
3     log.debug("Transaction details: " + txn.toString());
4     Coin value = txn.getValueSentToMe(wallet);
5     Pot currentPot = service.getCurrentPot();
6     if (currentPot.getExpectedBettingAmount() > value.getValue()) {
7         log.warn("Player did not pay enough.");
8         return;
9     }
10    List<Participant> participants =
        service.getPossibleParticipants();
11    NetworkParameters params = service.getAppKit().params();
12    for (TransactionOutput txnOutput : txn.getOutputs()) {
13        Address a = txnOutput.getAddressFromP2PKHScript(params);
14        String address = a.toString();
15        for (Participant participant : participants) {
16            String depositAddress = participant.getDepositAddress();
17            if (depositAddress.equals(address)) {
18                log.info("Received " + value.toFriendlyString() + " coins
                    from " + participant.toString());
19                participant.setReceivedAmount(value.getValue());
20                String fromAddress =
                    txn.getInput(0).getFromAddress().toString();
21                participant.setPayoutAddress(fromAddress);
22                wallet.addTransactionConfidenceEventListener(new
                    TxnConfidenceListener(service, txn, participant));
23            }
24        }
25    }
26 }

```

Zunächst wird in Zeile 6 geprüft, ob der eingegangene Zahlungsbetrag mindestens so hoch ist, wie vom aktuellen Topf gefordert. Ist dies der Fall, iteriert die Methode über die Output Adressen¹³ der Transaktion (Zeile 12) und prüft mit welcher

¹²Dies umfasst alle Adressen, die mittels der `bitcoinAppKit.wallet().freshAddress()` Methode erzeugt wurden.

¹³Eine detaillierte Beschreibung wie Bitcoin Transaktionen aufgebaut sind, liefert [30].

Einzahlungsadresse (Zeile 17) diese übereinstimmt. Handelt es sich bei der Output Adresse nicht um die Einzahlungsadresse eines Teilnehmers, wird diese ignoriert, da es sich um eine Wechselgeldadresse handeln muss¹⁴. Hat man den Teilnehmer identifiziert, wird aus der Transaktion die Auszahlungsadresse berechnet und dem Teilnehmer der empfangene Geldbetrag gutgeschrieben (Zeile 19-21). Da es sich um eine gültige, jedoch noch unbestätigte Transaktion handelt, wird der Teilnehmer erst nachdem die Transaktion in einen gültigen Block aufgenommen wurde, zum Topf hinzugefügt. Zeile 22 erzeugt einen TxnConfidenceListener, der diese Aufgabe übernimmt.

```

1  @Override
2  public void onTransactionConfidenceChanged(Wallet wallet,
      Transaction txn) {
3      if (txn.equals(targetTxn)) {
4          log.debug("onTransactionConfidenceChanged Tx: " +
              txn.getHash());
5          switch (txn.getConfidence().getConfidenceType()) {
6              case PENDING:
7                  // unconfirmed but should be included shortly
8                  break;
9              case BUILDING:
10                 // transaction is included in the best chain
11                 Pot currentPot = service.getCurrentPot();
12                 participant.setPotIndex(currentPot.getNbrOfParticipants());
13                 currentPot.addParticipant(participant);
14                 if (currentPot.isFull()) {
15                     service.closeCurrentPot(new Date());
16                 }
17                 wallet.removeTransactionConfidenceEventListener(this);
18                 break;
19             case IN_CONFLICT:
20                 log.warn("possible double spend of txn " +
                    txn.getHashAsString());
21                 break;
22             case DEAD:
23                 log.warn("txn " + txn.getHashAsString() + " won't confirm
                    unless there is another re-org");
24                 wallet.removeTransactionConfidenceEventListener(this);
25                 break;
26         }
27     }
28 }

```

Die `onTransactionConfidenceChanged` Methode wird jedes Mal aufgerufen, wenn dem SPV-Client neue Daten zur Transaktion vorliegen. BitcoinJ unterscheidet zwischen vier verschiedenen `ConfidenceTypes`:

- **PENDING:** Bedeutet, dass die Transaktion noch unbestätigt ist und der SPV-Client darauf wartet, dass er einen Block erhält, in dem die Transaktion enthalten ist.
- **BUILDING:** Bedeutet, dass die Transaktion bereits in die Blockchain aufgenommen wurde. Durch den Aufruf von `transaction.getConfidence()`

¹⁴Bei Bitcoin können die einer Adresse zugeschriebenen Währungseinheiten entweder ganz oder gar nicht ausgegeben werden. Daher ist es üblich eine vom Wallet kontrollierte Wechselgeldadresse in jeder Transaktion anzugeben.

.getAppearedAtChainHeight() kann man abfragen, wie tief die Transaktion bereits in der Blockchain steckt. Diese Methode gibt zurück, wie viele Blöcke bereits auf den Block, der die Transaktion enthält, aufbauen. Die Glücksspielanwendung betrachtet eine Transaktion als final, sobald sie in einen gültigen Block aufgenommen wurde.¹⁵ Geschieht dies, wird der Teilnehmer in den Topf hinzugefügt.

- IN CONFLICT: In diesem Fall hat der SPV-Client zwei Transaktionen erhalten, die versuchen den gleichen Transaction-Output auszugeben. Man spricht von einem sogenannten "double spend" Angriff.
- DEAD: Transaktionen, die diesen Status erhalten, können nicht mehr bestätigt werden, außer es kommt zu einer Blockchain-Restrukturierung.

Immer wenn der SVP-Node einen neuen Block empfängt, den er vorne an die Blockheader Kette anhängen kann, wird die notifyNewBestBlock Methode aufgerufen.

```

1  @Override
2  public void notifyNewBestBlock(StoredBlock block) throws
    VerificationException {
3      log.info("New Block height = " + block.getHeight() + " hash = "
4              + block.getHeader().getHash().toString());
5      List<Pot> unfinishedPots =
        getUnfinishedPots(service.getClosedPots());
6      for (Pot pot : unfinishedPots) {
7          long potId = pot.getCreateTime().getTime();
8          int payoutBlockHeight = pot.getPayoutBlockHeight();
9          if (payoutBlockHeight > block.getHeight()) {
10             log.info("Pot[" + potId + "] can not be handled yet.");
11         } else if (payoutBlockHeight == block.getHeight()) {
12             log.info("Pot[" + potId + "] select temorary Winner.");
13             Block tmpPayoutBlock = new
                ExtendedBlock(block.getHeader().getHash().toString());
14             pot.setPayoutBlock(tmpPayoutBlock);
15             selectWinner(pot, tmpPayoutBlock);
16         } else {
17             log.info("Pot[" + potId + "] select final winner.");
18             try {
19                 StoredBlock correctBlock =
                    getPastBlock(payoutBlockHeight, block);
20                 String payoutBlockHash =
                    correctBlock.getHeader().getHash().toString();
21                 Block finalPayoutBlock = new
                    ExtendedBlock(payoutBlockHash);
22                 pot.setPayoutBlock(finalPayoutBlock);
23                 Participant winner = selectWinner(pot,
                    finalPayoutBlock);
24                 log.info(winner.getDepositAddress() + " wins pot["
                    + potId + "].");
25                 startPayoutThread(pot);
26             } catch (BlockStoreException e) {
27                 log.info("Couldn't select final winner of Pot["
                    + potId + "]: " + e.getMessage(), e);

```

¹⁵Vor der Auszahlung des Gewinns kann die Anwendung erneut nachprüfen, ob es eine Restrukturierung der Blockchain durch einen Blockchain-Fork gab. Dadurch stellt die Anwendung sicher, dass alle Transaktionen Teil der längsten Blockkette geworden sind.

```

28         }
29     }
30 }
31 }

```

Diese Methode iteriert über alle bereits geschlossenen Töpfe für die noch keine Auszahlung stattgefunden hat. Sie unterscheidet dabei 3 Fälle:

1. Die Blocknummer des neusten Blocks ist kleiner als die Blocknummer, die den Topf entscheidet. In diesem Fall passiert nichts.
2. Der neue Block entscheidet den Topf, da die Blocknummer des Blocks gleich der PayoutHeight des Topfs ist. Der Gewinner des Topfs wird selektiert. Es findet allerdings keine Auszahlung statt. Die finale Auszahlung findet aus Sicherheitsgründen erst im nächsten Fall statt.
3. In diesem Fall gibt es mindestens einen Block, der auf dem Payout-Block des Topfs aufbaut. Ab diesem Zeitpunkt betrachtet die Anwendung den Gewinner als final.¹⁶ Der Gewinner des Topfs wird überschrieben und die Auszahlung wird in einem neuen Thread gestartet.

Die Klasse `ExtendedBlock` teilt den Blockhash zur Anzeige in die Werte `prefix`, `lastDigit` und `suffix` auf. Die Variable `lastDigit` speichert die letzte numerische Stelle des Blockhashs und wird zur Gewinnerauswahl verwendet.

```

1  public class ExtendedBlock extends Block {
2
3      private String prefix;
4      private String suffix;
5
6      public ExtendedBlock(String blockHash) {
7          super(blockHash, -1);
8          int position = blockHash.length() - 1;
9          while (position > 0) {
10             char c = blockHash.charAt(position);
11             int value = (int) c;
12             if (value >= 48 && value <= 57) { // numeric
13                 this.lastDigit =
14                     Integer.parseInt(String.valueOf(c));
15                 break;
16             }
17             position--;
18         }
19         this.prefix = blockHash.substring(0, position);
20         this.suffix = blockHash.substring(position + 1,
21             blockHash.length());
22     }
23 }

```

¹⁶An dieser Stelle kann man natürlich auch aus Sicherheitsgründen noch mehrere Blöcke abwarten, bevor die Anwendung eine Auszahlung startet. Ab einer Tiefe von 6 Blöcken gelten Bitcoin Zahlungen als irreversibel. Bei sehr hohen Zahlungen ist es empfehlenswert so lange abzuwarten.

Auszahlungen

Auszahlungen werden in einem eigenen Thread abgehandelt. Die Klasse `PayoutThread` ruft dazu die `payout` Methode auf.

```

1 private void payout(Pot pot) throws InsufficientMoneyException ,
   InterruptedException , ExecutionException {
2     if (pot.isPayoutStarted()) {
3         log.error("Payout already started: " +
4             pot.getPayoutTxnId() + " - " + pot.getPayoutError());
5     } else {
6         pot.setPayoutStarted(true);
7         Address winnerAddress = new
8             Address(bitcoinService.getNetworkParams(),
9                 pot.getWinner().getPayoutAddress());
10        Coin potValue =
11            Coin.SATOSHI.multiply(pot.getParticipants().size() *
12                pot.getExpectedBettingamount());
13        Wallet.SendResult result =
14            bitcoinService.getAppKit().wallet()
15                .sendCoins(bitcoinService.getAppKit().peerGroup(),
16                winnerAddress, potValue);
17        String txnId = result.tx.getHash().toString();
18        pot.setPayoutTxnId(txnId);
19        log.info("Payout TXN ID = " + txnId);
20        Transaction transaction = result.broadcastComplete.get();
21        log(transaction);
22    }
23 }

```

Die Methode prüft zunächst, dass die Auszahlung noch nicht gestartet wurde. Ist dies der Fall, wird der auszuzahlende Betrag berechnet und an die Adresse des Gewinners überwiesen. Anschließend wird die ID der Auszahlungstransaktion in den Topf geschrieben. Während der Auszahlung können 3 verschiedene Fehler auftreten. Diese werden von BitcoinJ durch die folgenden Exceptions repräsentiert:

1. `InsufficientMoneyException`: Falls die von der Wallet verwalteten Adressen nicht genügend Bitcoin für die Auszahlung besitzen.
2. `InterruptedException`: Falls der Java Thread unterbrochen wird.
3. `ExecutionException`: Falls es zu einem unerwarteten Fehler bei der Ausführung kommt.

Sollte es bei der Auszahlung ein Problem geben, wird die Exception gefangen und im `Pot` Objekt unter `payoutError` abgespeichert.

2.3.5 Grafische Benutzeroberfläche

Die graphische Oberfläche der Anwendung ist mit dem Tapestry Framework von Apache realisiert. Da die GUI Komponente nur die Daten visualisiert, wird an dieser Stelle auf eine genauere Betrachtung verzichtet und lediglich die Benutzeroberfläche gezeigt.

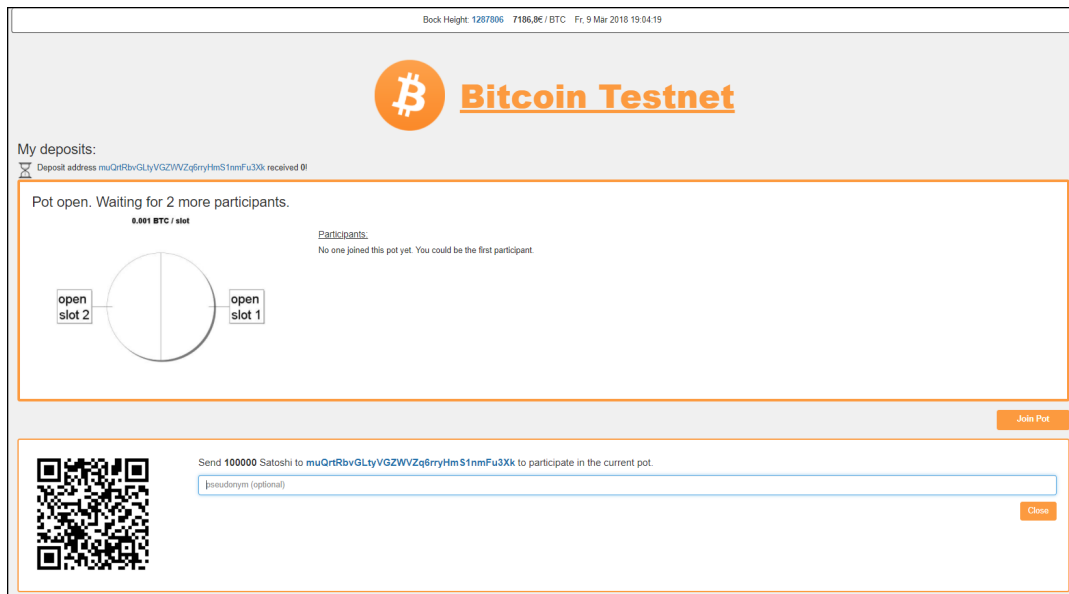


ABBILDUNG 2.21: Leerer Topf

Abbildung 2.21 zeigt einen Topf mit 2 freien Plätzen. Um dem Spiel beizutreten, muss der Spieler den Betrag von 0,001 Bitcoin an die angezeigte Adresse senden. Der angezeigte QR-Code erleichtert dem Spieler die Übertragung dieser Daten in das Überweisungsformular seines Smartphone Wallets. Das Bitcoin Improvement Proposal Nummer 21 [3] legt fest, in welchem Format diese Daten kodiert werden müssen, damit beliebige Bitcoin Clients diese auslesen und interpretieren können. Folgende Daten sind im QR Code enthalten:

"bitcoin:muQrtRbvGLtyVGZWVZq6rryHmS1nmFu3Xk?amount=0.001"

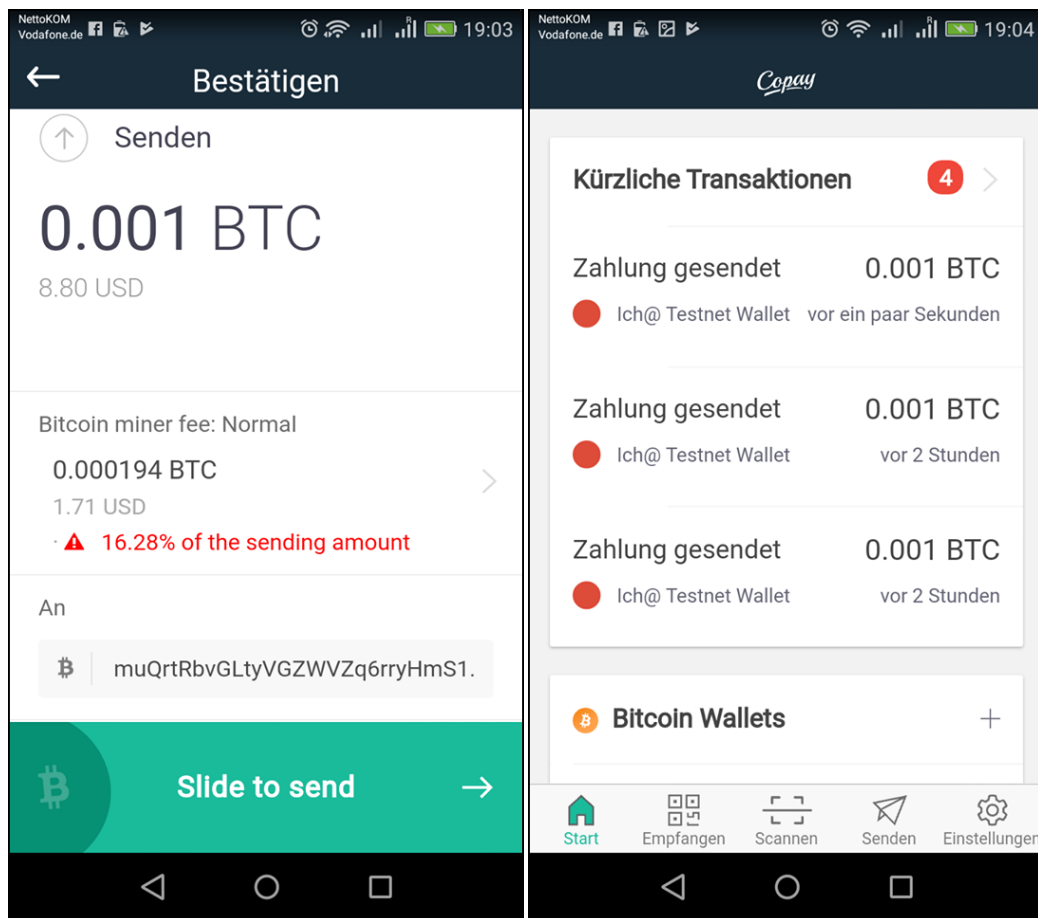


ABBILDUNG 2.22:
Smartphone Über-
weisungsformular

ABBILDUNG 2.23:
Zahlungsbestäti-
gung

Abbildungen 2.22 und 2.23 zeigen das Bitcoin CoPay Wallet¹⁷. Dieses erlaubt, neben Zahlungen im Bitcoin Hauptnetzwerk, auch Zahlungen an das Bitcoin Testnetz zu senden. Nachdem der Benutzer den QR-Code der Glücksspielanwendung mit seinem Smartphone abgescannt hat, erscheint sowohl der Betrag als auch die Empfangsadresse vorausgefüllt im Überweisungsformular. Die Wallet berechnet automatisch eine passende Transaktionsgebühr. Der Spieler überprüft lediglich die Adresse und den Betrag und autorisiert anschließend die Zahlung.

¹⁷<https://copay.io/>

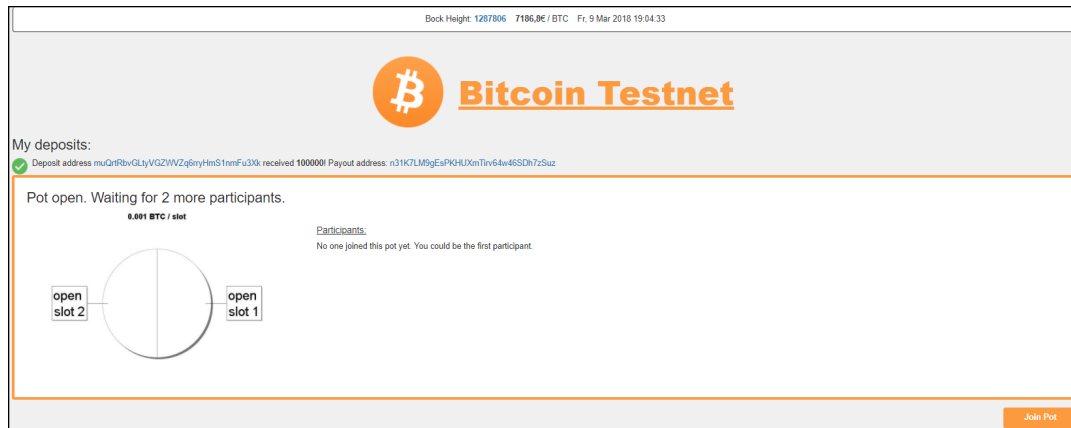


ABBILDUNG 2.24: Transaktion empfangen

Sobald die Anwendung die Transaktion empfängt, zeigt sie dies durch einen grünen Haken an. Dies ist in Abbildungen 2.24 zu sehen. Zu diesem Zeitpunkt handelt es sich um eine unbestätigte Transaktion, die noch in keinen Block der Blockchain aufgenommen wurde.

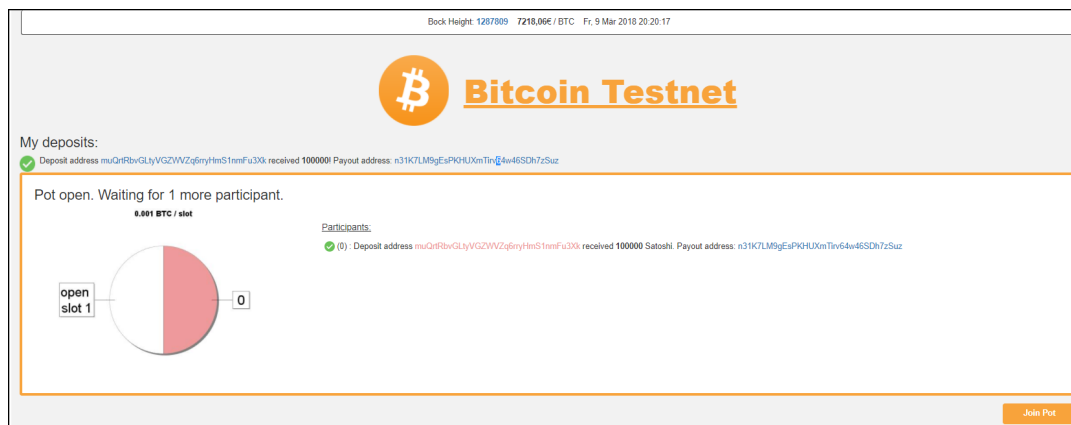


ABBILDUNG 2.25: Spieler zu Topf hinzugefügt.

Sobald die Anwendung einen neuen Block empfängt, der die Transaktion enthält, gilt die Transaktion als bestätigt und der Spieler wird zum Topf hinzugefügt. Nun gibt es nur noch einen offenen Platz im Topf.

My deposits:

Deposit address `muQrRbvGLjyVGZWZgfnYhmS1nmFu3Xk` received 100000!

Deposit address `n31K7LM5gEsPKHUXmTrnV64w46SDh7zSuz`

Pot open. Waiting for 5 more participants.

0.001 BTC / slot

open slot 5

open slot 4

open slot 1

open slot 2

open slot 3

Participants:

No one joined this pot yet. You could be the first participant.

Closed Pots

Pot closed. Waiting for block 1287811 to select the winner.

0.001 BTC / slot

1

0

Participants:

(0) Deposit address `muQrRbvGLjyVGZWZgfnYhmS1nmFu3Xk` received 100000 Satoshi.

(1) Deposit address `nDM9auRiATMBkAbYfXG6mQpdRbvCj6YjGc` received 100000 Satoshi.

ABBILDUNG 2.26: Topf geschlossen

Nachdem ein weiterer Spieler in den Topf eingezahlt hat, ist der Topf voll. Die Anwendung schließt den aktuellen Topf und öffnet einen neuen Topf. Die Anwendung erstellt zufällig entweder einen Topf mit 2, 5 oder 10 freien Plätzen. In einer real eingesetzten Glücksspielanwendung könnten alle Spielvarianten parallel angeboten werden. Der in Abbildung 2.26 gezeigte neue Topf hat fünf freie Plätze. Die Anwendung wartet nun auf den Payout-Block, um den Gewinner des geschlossenen Topfs festzulegen. Da der Blockhash noch nicht feststeht, zeigt die Anwendung die Animation eines sich sehr schnell wechselnden Blockhashs an, der mehrere Fragezeichen enthält.

Closed Pots

Pot closed. Winner is n4M9auRrATMBkAbY6X6xnGpxHkbvCdGYgC. Triggering payout!

0.001 BTC / slot

Winner calculation: 3 modulo 2 = 1 (hover for explanations)

	Block Height	Block Hash
Payout	1287811	0000000004597681397ad564fde4486a737cc909f7f35093894474d76a893
Closing	1287810	00000000a0d6c09f06d471495a584689e6d4efbe9f3f26f240154ab3c7ab446c

Participants:

- (0) : Deposit address `nuCqRbVGLyYQ2YMZg6yHwS1rmfX0X`, received 100000 Satoshi. Payout address: `n31KL7M5gASPKHXmTiv64w46SDN7zSuz`
- (1) : Deposit address `n4M9auRrATMBkAbY6X6xnGpxHkbvCdGYgC`, received 100000 Satoshi. Payout address: `mznU6NqkgPpsTfpaVZCwFyghw7ghaM8v`

ABBILDUNG 2.27: Gewinner ermittelt

Sobald der entscheidende Block empfangen wird, wird der voraussichtliche Gewinner des Topfs durch ein rotes Rechteck markiert. Außerdem zeigt die Anwendung durch welche Berechnung der Gewinner festgelegt wurde. Nun wartet die Anwendung bis ein weiterer Block gefunden wird, bevor sie die Auszahlung an den Gewinner startet.

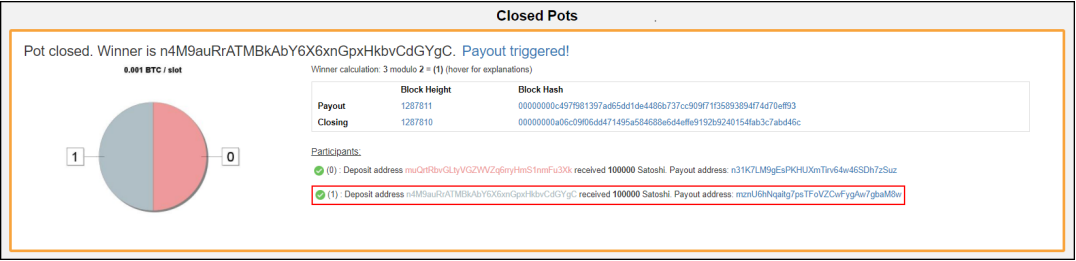


ABBILDUNG 2.28: Auszahlung beendet

Klickt der Benutzer auf den *Payout triggered* Link aus Abbildung 2.28, gelangt er zu einem Blockchain-Explorer. Dieser zeigt ihm die Transaktionsdetails an. Eine genauere Betrachtung findet in Abschnitt 2.4.5 statt.

2.4 Evaluation

2.4.1 Prüfung der Anforderungen

Dieser Abschnitt behandelt, in wieweit das beschriebene Konzept die in Abschnitt 1.4 aufgelisteten Anforderungen erfüllt. Die jeweilige Anforderung wird zunächst wiederholt und anschließend genauer untersucht.

1) Transparente Einzahlungen

Die Einzahlung jedes Endnutzers ist für jeden anderen Endnutzer nachprüfbar.

Diese Anforderung ist erfüllt, da jede Transaktion in der lokalen Datenbank jedes Peer-to-Peer Netzwerkteilnehmers aufgezeichnet wird. Auf der Webseite [5] kann man die Bitcoin Blockchain mithilfe eines sogenannten Blockchain-Explorers durchsuchen. Mit diesem Werkzeug kann man die Blöcke und die darin enthaltenen Transaktionen untersuchen. Nutzt man den Explorer einer Drittpartei, muss man darauf vertrauen, dass dieser auch den "wahren" Status der Blockchain anzeigt. Um dieses Risiko zu vermeiden, kann jeder Teilnehmer mithilfe eines eigenen Bitcoin Full Nodes am Netzwerk teilnehmen. Dieser speichert die gesamte Blockchain und prüft alle Transaktionen und Blöcke gegen die Konsensregeln.

Der Bitcoin Full Node stellt eine API bereit, über die man den aktuellen Status der Blockchain abfragen kann. Der Befehl `getblockchaininfo` liefert den aktuellen Zustand der Blockchain zurück. Dieser beinhaltet die Blocknummer des neusten Blocks und dessen Blockhash. Der Befehl `gettransaction` gefolgt von der Transaktions-ID liefert Details über diese Transaktion. Die Webseite [2] dokumentiert diese Schnittstelle detailliert.

2) Gewinnerauswahl durch Zufallsfaktor

Die Auswahl des Gewinners ist von einem zufälligen Faktor abhängig, auf den weder die Anwendung noch die Endnutzer einen Einfluss haben.

Diese Anforderung wird nur bedingt erfüllt, da ein Teilnehmer des Peer-to-Peer Netzwerks sowohl ein Spieler als auch ein Miner sein kann. Ist dies der Fall besteht die Möglichkeit, dass der Miner einen validen Blockhash verwirft, sobald er merkt, dass er durch diesen Blockhash nicht zum Gewinner des Geldtopfs wird. Verwirft der Teilnehmer einen Blockhash, riskiert er den dadurch ausgeschütteten Blockreward. Ein solcher Angriff ist für einen Miner nur rentabel, falls die Einnahmen des Glücksspiels hoch genug sind, um den potentiellen Verlust des Blockrewards zu kompensieren. Die Rentabilität eines solchen Manipulationsversuchs hängt außerdem von der Hashrate des Miners ab. Diese bestimmt mit welcher Wahrscheinlichkeit der Miner einen weiteren gültigen Block findet. Der Abschnitt 2.4.4 betrachtet dieses Szenario detailliert.

3) Nachprüfbarkeit des Zufallsfaktors

Jeder Endnutzer kann die Echtheit des zufälligen Faktors eigenständig nachprüfen.

Da das Verfahren der Gewinnerauswahl im Vorhinein festgelegt ist und die Reihenfolge der Einzahlungstransaktionen in der Blockchain festgeschrieben steht, kann jeder Teilnehmer die Berechnung des Gewinners eigenständig nachvollziehen.

Der Blockhash, der die Grundlage für die Gewinnerauswahl liefert, kann durch die Verwendung eines Blockchain-Explorers oder eines Full Nodes nachgeprüft werden.

4) Transparente Auszahlungen

Die Auszahlung an den Gewinner muss transparent und somit für jeden Endnutzer nachprüfbar sein.

Genau wie bei den Einzahlungen ist auch die Prüfung der Auszahlung für die Spieler mithilfe eines Blockchain-Explorers oder eines Bitcoin Full-Nodes möglich. Jeder Teilnehmer kann somit für alle bereits abgeschlossenen Spiele nachprüfen, ob die Anwendung sich korrekt verhalten und eine Auszahlung getätigt hat.

5) Fairheit des Spiels

Jeder Endnutzer besitzt die gleiche Gewinnwahrscheinlichkeit und niemand wird benachteiligt.

Die Zuordnung der Spieler auf die Gewinnzahlen ist durch die Reihenfolge der Transaktionen in der Blockchain festgeschrieben. Eine nachträgliche Veränderung dieser Reihenfolge ist weder durch die Nutzer, noch durch die Glücksspielanwendung möglich.¹⁸

Damit keiner der Spieler einen Vorteil hat, muss jeder Topf-Platz die gleiche Gewinnwahrscheinlichkeit haben. Dies ist gegeben, falls jeder Teilnehmer die gleiche Anzahl Gewinnzahlen zugeordnet bekommt und falls das Auftreten jeder Gewinnzahl die gleiche Wahrscheinlichkeit besitzt. Durch die Einschränkung der Topfgröße auf 2, 5 und 10 Teilnehmer bekommt jeder Spieler durch die Modulo-Funktion genau gleich viele Gewinnzahlen zugeordnet. Die Monte-Carlo-Simulation aus Abschnitt 2.4.3 legt nahe, dass die letzte Dezimalziffer der Werte gleichverteilt ist. Das Auftreten jeder Gewinnzahl ist somit gleich wahrscheinlich und die Anforderung erfüllt.

2.4.2 Gewinnerauswahl

Die Gewinnerauswahl kann entweder durch den gesamten Blockhash-Wert oder auf Basis der letzten Blockziffer vorgenommen werden. Beide Methoden haben Vor- und Nachteile. Variante eins erlaubt beliebige Topfgrößen, ist dafür aber schwieriger für den Endnutzer zu verifizieren. Die Verifizierung erfordert eine Modulo-Rechnung mit einer sehr großen Zahl. Variante zwei ist dagegen leicht zu verifizieren, erlaubt allerdings nur die Topfgrößen zwei, fünf und zehn. Bei der Topfgröße von zwei sind beiden Spielern fünf Gewinnzahlen zugeordnet. Bei der Topfgröße von fünf besitzt jeder Teilnehmer genau 2 Gewinnzahlen. Bei einer Topfgröße von zehn wird jedem Teilnehmer genau eine Gewinnzahl zugeordnet. Nimmt man hingegen eine Topfgröße von 3,4,6,7,8 und 9 führt dies dazu, dass manche Teilnehmer eine signifikant höhere Gewinnchance haben. Bei der Topfgröße von 3 sind die Gewinnzahlen durch die Modulo-Funktion folgendermaßen verteilt:

- Spieler 1 hat die Gewinnzahlen 0, 3, 6 und 9.
- Spieler 2 hat die Gewinnzahlen 1, 4 und 7.

¹⁸Eine Veränderung der Reihenfolge ist nur durch einen sogenannten Blockchain-Fork möglich. (siehe Abschnitt 2.4.7)

- Spieler 3 hat die Gewinnzahlen 2, 5 und 8.

Somit hat Spieler 1 eine Gewinnwahrscheinlichkeit von $\frac{4}{10}$, Spieler 2 und 3 hingegen nur eine Gewinnwahrscheinlichkeit von $\frac{3}{10}$. Es kommt also vor, dass eine Teilmenge der Spieler genau eine Gewinnzahl mehr als der Rest der Teilnehmer hat.

Nimmt man den gesamten Blockhash zur Gewinnerauswahl, ist die dadurch entstehende Ungerechtigkeit verschwindend gering und kann vernachlässigt werden. Dies ist der Fall, da die aus dem Blockhash resultierende Dezimalzahl in der Praxis sehr groß ist und jeder Spieler somit mehrere Millionen von Gewinnzahlen hat.

2.4.3 Verteilung der Blockhash-Werte

Der Blockhash eines Blocks wird durch die verwendete kryptographische Hashfunktion der Kryptowährung festgelegt. Diese bestimmt somit auch die Verteilung der Hashwerte. Eine kryptographische Hashfunktion ist eine stark kollisionsresistente Einweg-Hashfunktion.¹⁹

Eine Hashfunktion h heißt

- *Einwegfunktion* genau dann, wenn es schwierig ist, zu gegebenem y_0 ein x_0 zu finden mit $h(x_0) = y_0$.
- *schwach kollisionsresistent* genau dann, wenn es schwierig ist, zu einem gegebenen x ein $x' \neq x$ zu finden mit $h(x) = h(x')$.
- *stark kollisionsresistent* genau dann, wenn es schwierig ist, x und x' zu finden mit $x \neq x'$ und $h(x) = h(x')$.

Die Eigenschaften der starken Kollisionsresistenz und der Einwegfunktion sagen nichts über die Verteilung der resultierenden Werte aus. Bei der Auswahl der Kryptowährung muss also gesondert auf die Verteilung der verwendeten Hashfunktion geachtet werden. Sollte die verwendete kryptographische Hashfunktion keine Gleichverteilung liefern, kann der Blockhash dennoch den nötigen Zufall liefern indem dieser mit einer geeigneten Hashfunktion erneut gehasht wird.

Bitcoin verwendet die kryptographische Hashfunktion SHA256. Die folgende Monte-Carlo-Simulation legt nahe, dass die Resultate der SHA256 Hashfunktion gleichverteilt sind.

```
h=SHA256 n=1000000
for i 1 -> n
  hash = h(i);
  result[lastDigit(hash)]++
```

¹⁹Diese Definition samt der Klärung der Begriffe Einwegfunktion und schwach bzw. stark kollisionsresistent entstammt S.29 aus [25].

Ausgabe:

```
result[0] = 99765
result[1] = 100488
result[2] = 99913
result[3] = 100745
result[4] = 100272
result[5] = 99649
result[6] = 99430
result[7] = 99788
result[8] = 99666
result[9] = 100284
```

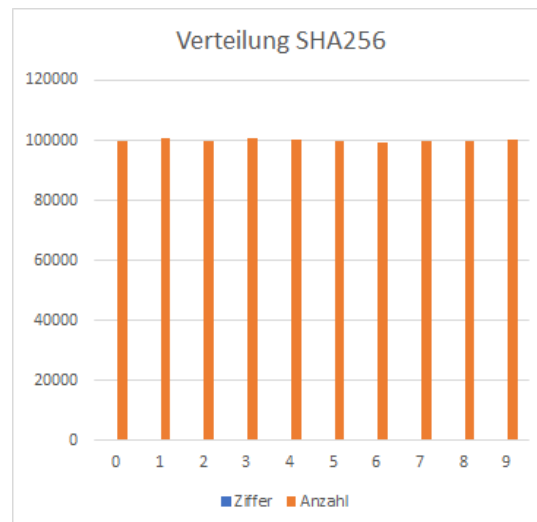


ABBILDUNG 2.29:
Verteilung der SHA256
Hashfunktion

2.4.4 Manipulationsversuch durch Miner

Dieser Abschnitt untersucht ab welchem Einsatzbetrag es für einen Miner profitabel ist, einen gültigen Blockhash zu verwerfen, um den Ausgang des Glücksspiels zu beeinflussen.

Betrachten wir dazu das Bitcoin Netzwerk Anfang Februar 2018. Der Preis pro Bitcoin beträgt 8000 Euro. Der Mining Reward liegt bei 12,5 Bitcoin pro Block²⁰. Für das Lösen eines gültigen Blocks erhält ein Miner somit 100000 Euro. Angenommen ein Miner besitzt eine Hashrate h und nimmt an einem Glücksspieltopf mit $n \in \{2, 5, 10\}$ Personen und einem Einsatz von E teil. Sollte ein anderer Miner des Bitcoin Netzwerks den Block für die Gewinnerauswahl finden, kann der Miner den Ausgang des Spiels nicht beeinflussen. Ausschließlich wenn der Miner einen gültigen Block vor dem Rest des Netzwerks findet und durch diesen das Spiel verliert, macht es für ihn Sinn, einen Manipulationsversuch in Betracht zu ziehen. Um eine Aussage über die Profitabilität machen zu können, muss der Erwartungswert für beide folgenden Szenarien berechnet werden:

Szenario 1

Der Miner akzeptiert sein Schicksal und veröffentlicht den Block. Es kommt zu keinem Manipulationsversuch. Der Erwartungswert beträgt $100000 - E$ Euro.

Szenario 2

Der Miner verwirft den Block und versucht sein Glück erneut, um den Ausgang des Spiels zu seinen Gunsten ändern zu können. Da es sich beim Mining um einen gedächtnislosen Prozess handelt, hat der Miner ab diesem Moment erneut eine Wahrscheinlichkeit von h einen gültigen Block zu finden. Für die Berechnung des Erwartungswerts müssen die folgenden Ausgänge berücksichtigt werden:

²⁰Dieses Beispiel vernachlässigt die durch das Minen des Blocks erhaltenen Transaktionsgebühren.

1. Fall: Der Miner findet einen weiteren Block, der das Spiel zu seinen Gunsten entscheidet. Dieser Fall tritt mit einer Wahrscheinlichkeit von $h * \frac{1}{n}$ ein. Der Miner erhält den Blockreward und den Einsatz der Mitspieler: $100000 + (n - 1) * E$ Euro.

2. Fall: Der Miner findet zwar einen weiteren Block doch verliert durch diesen erneut das Spiel. Dieser Fall tritt mit einer Wahrscheinlichkeit von $h * \frac{n-1}{n}$ ein. In diesem Fall entscheidet er sich den Block zu veröffentlichen, um den Blockreward zu erhalten. Der Miner erhält somit $100000 - E$ Euro.²¹

3. Fall: Ein anderer Miner findet den nächsten Block, der das Glücksspiel zu Gunsten des Miners entscheidet. Dieser Fall tritt mit einer Wahrscheinlichkeit von $(1 - h) * \frac{1}{n}$ ein. Der Miner verliert den Blockreward, gewinnt allerdings das Spiel und erhält somit $(n - 1) * E$ Euro.

4. Fall: Ein anderer Miner findet den nächsten Block und ein anderer Teilnehmer gewinnt durch diesen das Spiel. Dieser Fall tritt mit einer Wahrscheinlichkeit von $(1 - h) * \frac{n-1}{n}$ ein. Der Miner verliert den Blockreward und seinen Einsatz: $-E$ Euro.

Der Erwartungswert von Szenario 2 berechnet sich durch:

$$h * \frac{1}{n} * (100000 + (n - 1) * E) + h * \frac{n-1}{n} * (100000 - E) + (1 - h) * \frac{1}{n} * (n - 1) * E + (1 - h) * \frac{n-1}{n} * (-E)$$

Durch Vereinfachen ergibt sich ein Erwartungswert von $h * 100000$ Euro.²²

Setzt man die Erwartungswerte beider Szenarien gleich, ergibt sich folgende Gleichung: $100000 - E = h * 100000$.

Formt man diese nach E um, ergibt sich: $E = 100000 * (1 - h)$.

Diese Gleichung gibt an wie hoch der Einsatz des Spiels sein muss, damit es für einen Miner mit Hashrate h finanziell sinnvoll ist, das Spiel durch Zurückhalten eines gültigen Blocks zu manipulieren.

Der folgende Abschnitt betrachtet in welche Größenordnung h in der Praxis zu erwarten ist. Die gesamte Hashrate des Bitcoin Netzwerks beträgt zurzeit circa 30000000 Tera-Hash pro Sekunde (TH/s) [36]. Einer der profitabelsten Bitcoin ASICs ist der Antminer S9 der Firma Bitmain²³. Dieser kostet im Einkauf 900 Euro und liefert eine Rechenleistung von 14 TH/s. Ein Miner der ein zwanzigstel ($h = 0,05$) der Rechenleistung des Bitcoin Netzwerks besitzt, benötigt umgerechnet über 100000 Antminer S9. Somit betragen alleine die Anschaffungskosten der Mining Hardware 90 Millionen Euro.

Bei Proof-of-Work Kryptowährungen wie Bitcoin hat es sich durchgesetzt, dass Miner ihre Rechenleistung in Mining Pools bündeln und gemeinsam nach dem nächsten Block suchen. Der Mining Pool stellt dabei eine zentrale Instanz dar, die den nächsten Block zusammenstellt und dessen Blockheader an alle Teilnehmer verteilt. Die Teilnehmer empfangen den Blockheader und beginnen mit der Suche nach einem gültigen Blockhash. Findet der Pool einen gültigen Block, wird der Blockreward unter den Teilnehmern des Pools aufgeteilt. Üblicherweise erhalten die Betreiber des Mining Pools einen Anteil von 1 bis 2 Prozent für die Bereitstellung ihres Services.

²¹Der Miner könnte den Block erneut verwerfen und versuchen einen weiteren gültigen Block zu finden.

²²Die Anzahl Teilnehmer des Spiels hat keine Auswirkung auf den Erwartungswert. Die Variable n verschwindet beim Vereinfachen der Erwartungswertgleichung.

²³<https://www.bitmain.com/>

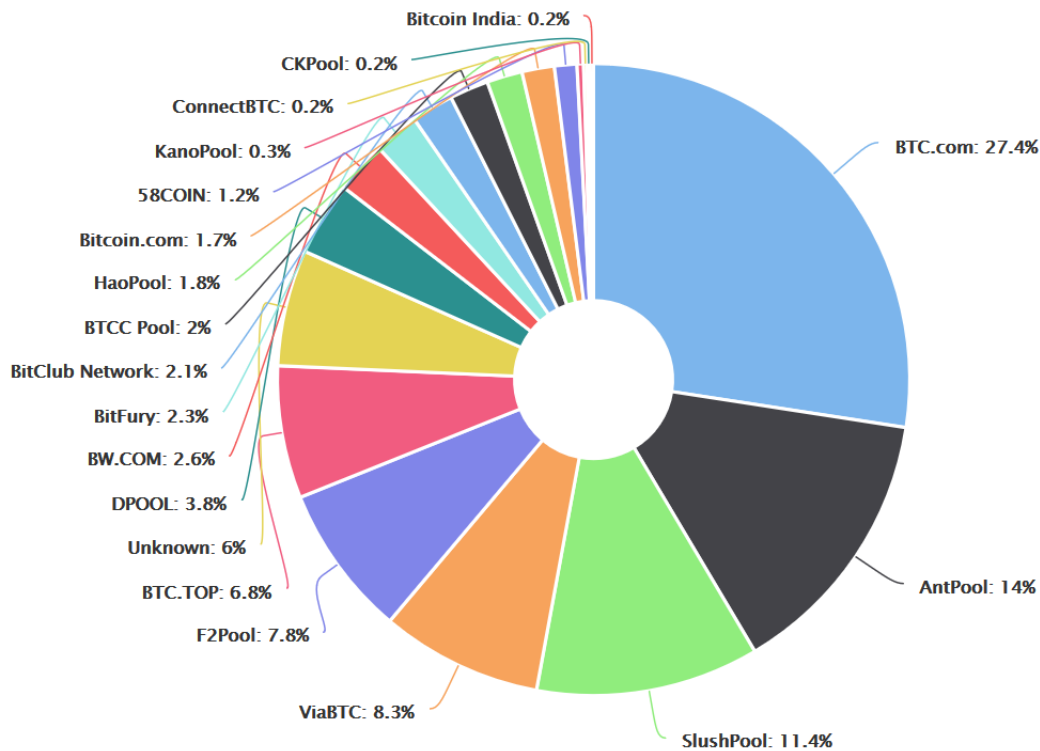


ABBILDUNG 2.30: Bitcoin Mining Pool Hash Rate [37]

Abbildung 2.30 zeigt die Verteilung der Hashrate des Bitcoin Netzwerks basierend auf den Blöcken der 4 letzten Tage.²⁴ Aus der Grafik geht hervor, dass der größte Pool circa 25% der Hashrate des Bitcoin Netzwerks besitzt. Hierbei ist es wichtig hervorzuheben, dass sich die eigentliche Mining Hardware im Besitz von Privatpersonen befindet. Diese Privatpersonen können den Mining Pool wechseln, sollten sie nicht mit den Praktiken des Pools einverstanden sein. Ein Pool, der gültige Blöcke nicht an das Bitcoin Netzwerk weiterleitet, da er den Ausgang eines Glücksspiels bestimmen möchte, riskiert damit Marktanteile einzubüßen.

²⁴Die Graphik wurde am 10.05.2018 auf Blockchain(dot)Info erstellt. Der Bitcoin Knoten von Blockchain(dot)Info ist mit den aufgelisteten Pools verbunden und schreibt einen neuen Block dem Pool zu, von dem Blockchain(dot)Info diesen als erstes empfangen hat. Die Statistik ist daher mit Vorsicht zu genießen.

Die oben berechnete Formel $E = 100000 * (1 - h)$ ergibt die in Abbildung 2.31 gezeigte Gerade.

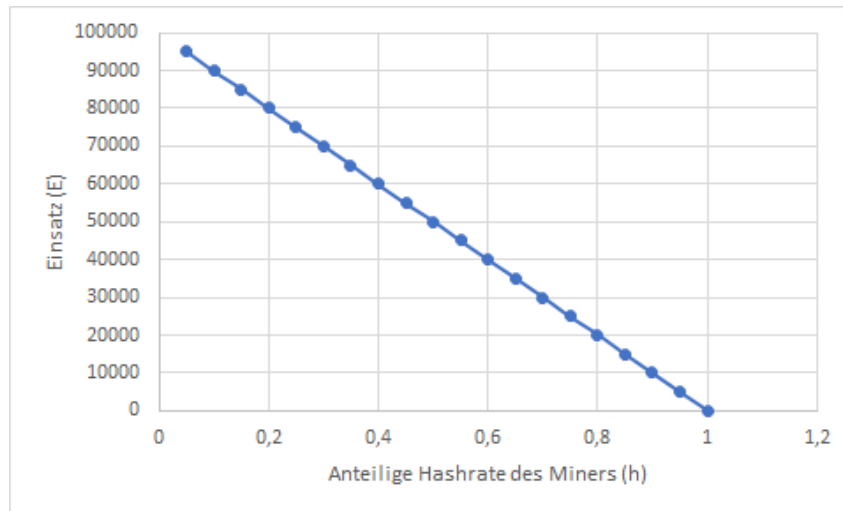


ABBILDUNG 2.31: Gleichung: $E = 100000 * (1 - h)$

Es geht hervor, dass ein Manipulationsversuch für einen Miner mit einem zwanzigstel der Hashrate ($h = 0,05$) erst finanziell lohnenswert ist, falls der Einsatz des Spiels über 95000 Euro liegt. Würde der größte Bitcoin Mining Pool ($h = 0,25$) einen Angriff starten, wären Töpfe bis zu einem Einsatz von 75000 Euro als sicher anzusehen. Selbst für einen Miner, der die Hälfte der Hashrate des Bitcoin Netzwerks ($h = 0,5$) kontrolliert, ist es nur profitabel gültige Blöcke zu verwerfen, falls der Einsatz des Spiels über 50000 Euro beträgt.

2.4.5 Auszahlungstransaktion

Die Glücksspielanwendung erzeugt für jede Einzahlung eine eigene Einzahlungsadresse statt für jeden Benutzer die gleiche Adresse zu verwenden. Dies hat den Vorteil, dass die Anwendung dem Benutzer dadurch anzeigen kann, dass seine Bitcoin Einzahlung eingegangen ist. Die folgenden Abbildungen betrachten die Auszahlungstransaktion des Beispiels aus Abschnitt 2.3.5.

Transaction Details (₿)

554924df2b8ba13bc540ffb903074cc41460390fd621f9d722759925de0520f6

ID	554924df2b8ba13bc540ffb903074cc41460390fd621f9d722759925de0520f6
Block No.	Unassigned
Coin	Bitcoin Testnet (BTC TEST, BT)
Time	Mar 9, 2018 at 03:40
Status	Unconfirmed
Confidence	0%
Confirmations	0
# of Inputs	3
# of Outputs	2
Sent Value	0.00300000
Fee	0.00052500
Other Info	Size: 520 bytes, Raw Data

ABBILDUNG 2.32: Auszahlungstransaktion Details

Abbildung 2.32 zeigt in welchen Block die Transaktion aufgenommen wurde, den Status, den Wert, die Transaktionsgebühr und die Größe der Transaktion.²⁵

Inputs / Senders			
Index	Address	Value (BT)	
< 0	mgYNmyrJExWoFRP4oGVM5Dt9BbZiqfAmBW	0.00100000	
< 1	mnnSU1EzgPgva3eFep8x2sNunoYCG71YsK	0.00100000	
< 2	mvGPrihYSaHz22XCmH78Cy97RSLhWUq6Xj	0.00100000	
		0.00300000	

Outputs / Receivers			
Index	Address	Value (BT)	
0	mznU6hNqaitg7psTFoVZCwFygAw7gbaM8w	0.00200000	>
1	n3MSpm7PhCLDx3CfmWWF55n8jQMRjprWv	0.00047500	>
		0.00247500	

ABBILDUNG 2.33: Auszahlungstransaktion Inputs und Outputs

²⁵Momentan zahlt die Glücksspielanwendung die Auszahlungstransaktionsgebühr aus eigener Tasche. Eigentlich müsste die Transaktionsgebühr vom Gewinnbetrag abgezogen werden.

Abbildung 2.33 zeigt welche Inputs und Outputs für die Transaktion verwendet wurden. Output Adresse 1 gehört der Wallet der Glücksspielanwendung und stellt die Wechselgeldadresse dar.

Input, Output Scripts	
Index	Script
Input 0	30440220170de6082da75e45ca88323ffc378efa7006ecbde8d666134b3aca3c644e343602205f02b0ea117cbdee5722b6cc8415bac8db0d8f185d1ebb839fc22f08bdef7053010237d4f81b1bc5c115d719126e14992456cd6b5b9d1b807f38fd641fb7a8ebfe8b
Input 1	30450221009602c4995821fdbaf6e0810d7e86b49efa49863865f9c3a09612722486476aad02205530dcd679d6025aa2dbfcc6a89fd93b415a1cea84901a55dc0f1811d209fd0701033ab2c32ea895b8a72756127b4d70ce0a4d0a6f2b8de08007ee98cb1b388674d7
Input 2	304402203b71208e03fe7dc9af6b4f7f928198b93ee4427ab7a8b8417f8d9d3088a9cd1c02203d3036e9b532ce05f96992b52e348b736463ea8d872045f2af229ec029d874980103c2151045e755d42a8e26820d18b47767f047133412bcef34d5270a8fe8482f9e
Output 0	OP_DUP OP_HASH160 d3598233c4436478702e9f9b91f98b4fd3b6464e OP_EQUALVERIFY OP_CHECKSIG
Output 1	OP_DUP OP_HASH160 ef866c3c9608fa1785922c94c30e14514be7bf81 OP_EQUALVERIFY OP_CHECKSIG

ABBILDUNG 2.34: Auszahlungstransaktion Skripts

Die Anwendung muss in der Auszahlungstransaktion für jede Input Adresse eine gültige Signatur angeben, obwohl alle Adressen von der gleichen Wallet kontrolliert werden. Hier könnte in Zukunft die Verwendung sogenannter Schnorr Multi-Signaturen [41] aushelfen. Durch diese lassen sich alle Signaturen der Inputs durch eine einzige Signatur ersetzen.

2.4.6 Blockchain Mining Varianz

Der Begriff Blockchain Mining Varianz beschreibt den Umstand, dass die Miner des Netzwerks entweder Glück oder Pech bei der Suche nach dem nächsten gültigen Blockhash haben können. Bei Bitcoin gibt es daher nicht genau alle 10 Minuten einen neuen Block, sondern durchschnittlich alle 10 Minuten. Für die Glücksspielanwendung bedeutet dies, dass die Zeit zwischen der letzten Einzahlung und der Auswahl des Gewinners variieren kann. In der Praxis kommt es vor, dass man 30 Minuten und mehr auf den nächsten Block warten muss. Dies ist für den Spieler eine recht lange Zeit. Das Proposal [23] liefert ein Verfahren, das die Blockchain Mining Varianz stark verringert. Dieser Vorschlag ist bisher allerdings noch nicht in den Bitcoin Sourcecode eingeflossen. Eine andere Möglichkeit die Wartezeit für den Spieler zu verringern ist es eine Kryptowährung mit einer geringeren Blockzeit zu verwenden. Beispiele hierfür sind LiteCoin (2,5 Minuten), die auf Privatsphäre spezialisierte Währung Monero (2 Minuten) und Ethereum (12 Sekunden). Bei einer geringen Blockzeit kommt es häufiger zu sogenannten Blockchain-Forks. Weitere Informationen zu den genannten Kryptowährungen sind unter [20], [34] und [16] verfügbar.

2.4.7 Blockchain Forks

Blockchain Forks entstehen, falls zwei Miner unabhängig voneinander mehr oder weniger gleichzeitig einen validen Block finden. Beide Miner broadcasten ihren Block schnellstmöglich an die Teilnehmer des Peer-to-Peer Netzwerks. Aufgrund von Netzwerkverzögerungen kommt es nun dazu, dass ein Teil des Netzwerks Block 1 und der restliche Teil des Netzwerks Block 2 zuerst enthält. Beide Blockchain-Ketten sind nun gleich lang.

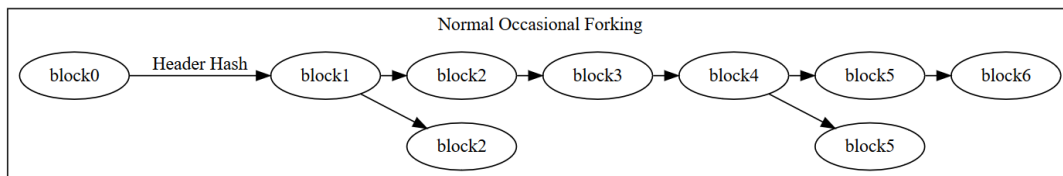


ABBILDUNG 2.35: Bitcoin Fork

Der nächste gefundene Block entscheidet, auf welche Kette sich das Netzwerk einigt²⁶. Die Bitcoin Konsensregeln legen fest, dass die Teilnehmer des Netzwerks immer der längsten Kette, die somit am meisten Proof-of-Work beinhaltet, folgen. Dies erlaubt es jedem Bitcoin-Knoten, ohne Trusted Third Party festzustellen, welche Version der Blockchain die Richtige ist. Forks kommen bei Bitcoin durch die hohe Hashrate des Netzwerks und die somit sehr hohe Difficulty recht selten vor. Die Webseite [29] zeigt an, wie oft gültige Blöcke gefunden und nicht Teil der längsten Blockchain werden.

2.4.8 Betrugsmöglichkeiten

Dieser Abschnitt betrachtet in wieweit die Glücksspielanwendung potentielle Spieler betrügen kann.

Die Anwendung hat die volle Kontrolle darüber welche Ausgabe sie dem Benutzer anzeigt. Sie hat allerdings keine Kontrolle über den Status der Blockchain. Die Anwendung könnte beispielsweise anzeigen, dass der Topf nach der Einzahlung durch einen Spieler immer noch leer ist. Die Transaktion auf die Einzahlungsadresse existiert dann zwar in der Blockchain, allerdings reagiert die Anwendung nicht entsprechend. Dies hat zur Folge, dass jeder Spieler, der eine Einzahlung tätigt, sein Geld verliert. Ein solch plumper Manipulationsversuch fällt dem Benutzer auf und er kann eine weitere Verwendung der Anwendung unterlassen. Ein weitere Betrugsmöglichkeit ist, dass die Glücksspielanwendung sich bis zur Gewinnerauswahl korrekt verhält, dann allerdings keine Auszahlung tätigt. Alle einzahlenden Spieler merken den Betrug, verlieren aber dennoch ihr Geld.

Bei den vorgestellten Betrugsversuchen fällt der Betrug immer mindestens einem Spieler auf. Die Verwendung einer solchen Anwendung macht nur Sinn, falls man den Betreiber des Services kennt und diesen im Zweifel juristisch haftbar machen kann.

Das folgende Kapitel beschreibt, wie sogenannte Smart Contracts dieses Problem lösen. Ein Smart Contract erlaubt es die Geschäftslogik der Glücksspielanwendung in die Blockchain zu schreiben. Die Geschäftslogik wird somit nicht mehr von der Anwendung, sondern von jedem Teilnehmer des Peer-to-Peer Netzwerks ausgeführt.

²⁶Unter der Annahme, dass es nicht erneut zu einem Blockchain Fork kommt.

Kapitel 3

Zweiter Ansatz: Ethereum

3.1 Grundlagen

Ethereum ist genau wie Bitcoin ein Peer-to-Peer Netzwerk, das eine öffentlichen Blockchain besitzt und einen Systemzustand verwaltet. Es handelt sich im Gegensatz zu Bitcoin um eine generalisierte Blockchain, die nicht nur Finanztransaktionen speichern kann, sondern auch sogenannte Smart Contracts¹. Ethereum sieht sich als eine Open Source Plattform, die es ermöglicht dezentrale Applikationen zu entwickeln und bereitzustellen. Die folgenden Begriffe und Informationen entstammen dem offiziellen Ethereum Whitepaper [6].

3.1.1 Smart Contracts

Ethereum ermöglicht es Geschäftsprozesse in Form von Smart Contracts zu programmieren und von einem globalen, dezentralen Netzwerk ausführen zu lassen. Ein Contract ist ein Programm, das aus einer Reihe von Anweisungen besteht, die ausgeführt werden, sobald das Programm eine Nachricht in Form einer Transaktion erhält. Contracts haben die Möglichkeit Daten aus der Blockchain auszulesen und auf ihr Daten zu speichern. Smart Contracts können sowohl von Menschen als auch von anderen Contracts ausgelöst werden. Contracts können daher mit anderen Contracts über die vom Programmierer festgelegte Schnittstelle interagieren. Es handelt sich bei einem Smart Contract also nicht um einen Vertrag, der erfüllt werden muss, sondern um ein öffentliches Stück Software, das den Zustand eines Ethereum Accounts verwaltet.

3.1.2 Ethereum Accounts

Um Ethereum nutzen zu können, braucht man einen Account. Ethereum Accounts bestehen aus:

- Einer 20 Byte langen Adresse
- einem Kontostand in Ether (siehe Abschnitt 3.1.5)
- einem Nonce-Wert²
- Smart Contract Code (optional)

¹Bitcoin Transaktionen beinhalten im Grunde auch Smart Contracts. Diese sind allerdings wesentlich weniger mächtig. Abschnitt 3.1.9 geht genauer auf die Unterschiede zwischen Bitcoin und Ethereum ein.

²Es handelt sich um einen dynamischen Wert, der hochgezählt wird und sicherstellt, dass Transaktionen nur einmal ausgeführt werden.

- Speicherplatz (optional)

In Ethereum gibt es zwei Arten von Accounts. Die erste Art von Account ist eine Adresse, die durch den Besitz eines privaten Schlüssels kontrolliert wird. Diese Art von Account wird, genau wie bei Bitcoin, durch eine Wallet Software verwaltet. Die zweite Art ist ein Contract Account, der durch den Smart Contract Code kontrolliert wird. Der Smart Contract Code verwaltet eigenständig den Kontostand des Accounts und verhält sich genau so, wie der Programmierer es festgelegt hat.

3.1.3 Transaktionen

Die Interaktion mit dem Ethereum Netzwerk findet in Form sogenannter Transaktionen statt. Transaktionen sind signierte Datenpakete, die eine Nachricht(siehe Abschnitt 3.1.4) beinhalten. Transaktionen beinhalten:

- den Empfänger der Nachricht (Ethereum Account Adresse)
- eine digitale Signatur, die den Absender der Nachricht identifiziert
- eine Anzahl an Ether, die an den Empfänger versandt wird
- Daten (Bytes), die vom Contract ausgelesen werden können (optional),
- einen sogenannten Gas-Wert, der die maximale Anzahl Rechenschritte festlegt, die die Transaktion bei der Ausführung nutzen darf
- den GAS-Preis, der festlegt wie viel der Absender bereit ist für die Ausführung eines Rechenschritts der Transaktion zu zahlen

Es gibt zwei verschiedene Arten von Transaktionen. Sie unterscheiden sich durch die in der Transaktion angegebene Empfangsadresse. Wird die Empfangsadresse durch einen privaten Schlüssel kontrolliert, handelt es sich lediglich um eine Finanztransaktion, die Ether vom Sender zum Empfänger Account transferiert. Anderenfalls handelt es sich um eine Transaktion, die den Contract Code der Empfangsadresse ausführt. Welche Funktion ausgeführt werden soll, wird vom Sender im Datenfeld der Transaktion festgelegt.

3.1.4 Nachrichten

Bei Nachrichten handelt es sich um virtuelle Objekte, die im Gegensatz zu Transaktionen nicht abgespeichert werden, sondern lediglich in der Ethereum Virtual Machine (siehe Abschnitt 3.1.6) verwendet werden. Ein weiterer Unterschied ist, dass Transaktionen im Gegensatz zu Nachrichten immer von "außen" erzeugt werden. Nachrichten können hingegen auch von Smart Contracts erzeugt werden. Nachrichten bestehen aus einem impliziten Sender, einem Empfänger, einem Ether-Betrag, einem Gas-Wert und einem optionalen Datenfeld. Genau wie bei Transaktionen führt das Versenden von Nachrichten zur Ausführung des Contract Codes der Empfangsadresse.

3.1.5 Ether

Ether ist die Kryptowährung des Ethereum Netzwerks. Sie entsteht wie bei Bitcoin durch Mining und wird benutzt, um für Transaktionsgebühren und die Ausführung von Contract Code zu bezahlen. Die kleinste Einheit der Ethereum Währung nennt sich WEI. Ein Ether entspricht 10^{18} WEI. Ether unter die Zuhilfenahme des Gas-Werts und des Gas-Preises den Anti-Spam Schutz des Ethereum Netzwerks.

Gas-Wert

Da Smart Contracts Schleifen beinhalten und andere Smart Contracts durch Nachrichten aufrufen können, muss sichergestellt werden, dass die Ausführung terminiert. Da es für den Sender der Transaktion schwierig (in manchen Fällen sogar unmöglich³) ist, die Anzahl Schritte der Ausführung im Vorhinein zu wissen, legt der Absender durch den Gas-Wert die maximale Anzahl Rechenschritte fest, die die Transaktion bei der Ausführung dauern darf.

Gas-Preis

Durch den Gas-Preis gibt der Sender einer Transaktion an, wie viel Ether er pro Rechenschritt bereit ist zu bezahlen. Die Ausführung einer Transaktion kann einen Ethereum Account somit um maximal $\text{Gas-Wert} * \text{Gas-Preis}$ Ether belasten.

3.1.6 Ethereum Virtual Machine

In der Ethereum Virtual Machine wird der Smart Contract Code ausgeführt. Wird ein Smart Contract durch eine Transaktion aufgerufen, arbeitet sie die Anweisungen des Contracts der Reihe nach ab und verändert dadurch den Zustand des Ethereum Accounts. Reicht der Gas-Wert der Transaktion nicht für die Ausführung aller Anweisungen aus, bricht die EVM ab und macht die Anpassungen des Systemzustands (bis auf die Zahlung der Transaktionsgebühr) rückgängig. Durch das sequenzielle Abarbeiten der Transaktionen eines Blocks wird der Systemzustand nach und nach von der Ethereum Virtual Machine angepasst.

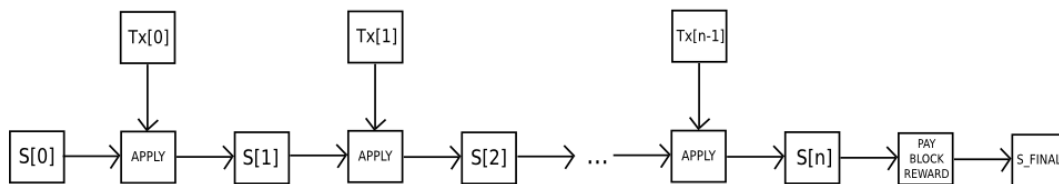


ABBILDUNG 3.1: Sequenzielle Veränderung des Systemzustands [6]

Der Systemzustand wird bei Ethereum durch den Zustand aller Ethereum Accounts beschrieben.

3.1.7 Systemzustand Übergangsfunktion

Die Ethereum Virtual Machine geht bei der Abarbeitung einer Transaktion folgendermaßen vor:

1. Sie prüft, ob die Transaktion gemäß der Konsensregeln korrekt formatiert ist, eine valide Signatur besitzt und ob der Nonce-Wert der Transaktion mit dem Nonce-Wert des Absender-Accounts übereinstimmt. Ist eine dieser Vorbedingungen nicht erfüllt, wird die Bearbeitung der Transaktion abgebrochen.

³Beispielsweise wenn der nächste Block eine Transaktion enthält, die eine Zustandsänderung bewirkt, sodass von der eigentlichen Transaktion ein unerwarteter Code-Strang durchlaufen wird.

2. Es wird die maximal fällige Transaktionsgebühr durch das Multiplizieren des Gas-Werts mit dem Gas-Preis berechnet. Anschließend wird geprüft, ob der Account des Senders genug Ether besitzt, um die berechnete Transaktionsgebühr zu bezahlen. Ist dies der Fall wird die maximale Transaktionsgebühr vom Sender-Account abgezogen. Anderenfalls wird die Bearbeitung der Transaktion abgebrochen.
3. Nun wird der Gas-Wert der Transaktion um einen gewissen Betrag für jedes Transaktionsbyte verringert. Der Sender zahlt auf diese Weise für den Platz, den die Transaktion in der Blockchain einnimmt.
4. Der Ether Wert der Transaktion wird vom Sender auf den Empfänger Account überwiesen. Wenn der Empfänger Account noch nicht existiert, wird er angelegt. Falls der Empfänger Account durch einen Smart Contract verwaltet wird, wird der Contract Code entweder vollständig ausgeführt oder aufgrund fehlenden Gases abgebrochen.
5. Falls der Sender nicht genug Ether oder verbleibendes Gas für die Überweisung besitzt, werden alle bisherigen Veränderungen des Systemzustands rückgängig gemacht. Die Transaktionsgebühr wird auf den Account des Miners überschrieben.
6. Im erfolgreichen Fall werden die Gebühren des verbleibenden Gas an den Sender zurückerstattet und die Gebühren des verbrauchten Gas an den Account des Miners überschrieben.

3.1.8 Systemzustand Beispiel

Das folgende Beispiel betrachtet wie das Anwenden einer Transaktion einen alten Systemzustand in einen neuen Systemzustand überführt.

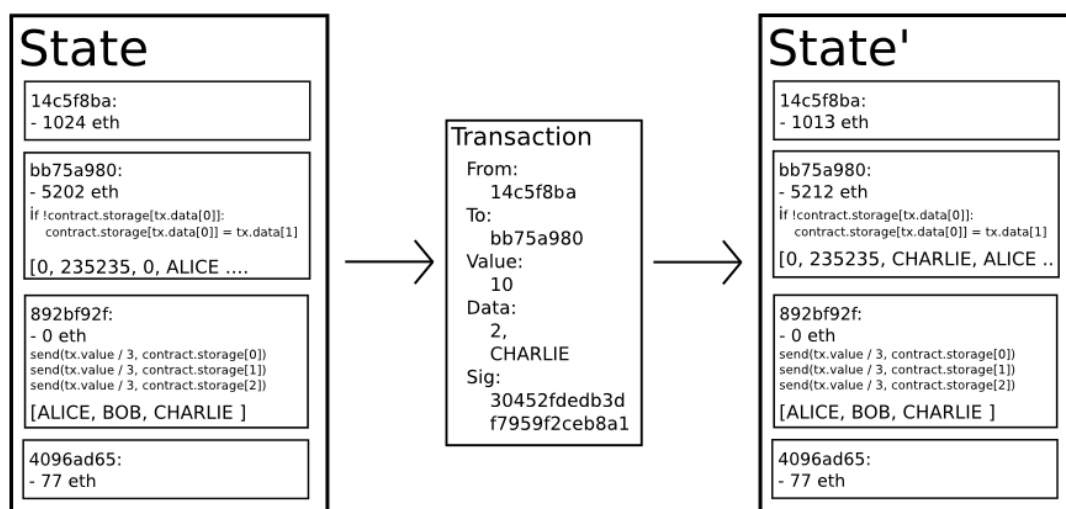


ABBILDUNG 3.2: Veränderung des Systemzustands Beispiel [6]

Die in Abbildung 3.2 betrachtete Transaktion ruft den Smart Contract des Accounts der Adresse bb75a980 auf. Angenommen die Transaktion überweist einen Betrag von 10 Ether, besitzt einen Gas-Wert von 2000, einen Gas-Preis von 0,001 Ether, enthält 64 Bytes an Daten(data[0]=2,data[1]='CHARLIE') und besitzt eine

Gesamtgröße von 170 Bytes. Dann erfolgt die Abarbeitung der Transaktion folgendermaßen:

1. Es wird durch die Prüfung der Signatur einerseits sichergestellt, dass die Daten der Transaktion nicht manipuliert wurden und andererseits, dass die Überweisung der 10 Ether vom Besitzer des Accounts 14c5f88a autorisiert wurde.
2. Da der Account des Senders über $2000 * 0,001 = 2$ Ether besitzt, wird die Bearbeitung der Transaktion fortgesetzt und die 2 Ether vom Account des Senders abgezogen.
3. Unter der Annahme, dass man 5 Gas pro Transaktionsbyte bezahlen muss, wird der Gas-Wert von 2000 auf 1150 Gas verringert. ($170 \text{ Bytes} * 5 = 850 \text{ Gas}$)
4. Der Transaktionsbetrag von 10 Ether wird vom Sender Account 14c5f88a auf den Account bb75a980 überwiesen.
5. Nun wird der Contract Code des Empfängeraccounts ausgeführt:

```
if !contract.storage[tx.data[0]]: contract.storage[tx.data[0]] = tx.data[1]
```


Da der Speicherplatz des Contracts an Stelle 2 noch leer ist, wird der Wert CHARLIE aus den Transaktionsdaten abgespeichert. Unter der Annahme, dass diese Operationen 150 Gas verbrauchen, ergibt sich ein verbleibender Gas-Wert von $1150 - 150 = 1000$.
6. Die verbleibenden $1000 * 0,001 = 1$ Ether werden auf den Account des Senders zurücküberwiesen und die Anpassung des Systemzustands ist fertig⁴.

3.1.9 Unterschiede zu Bitcoin

Turing-Vollständigkeit

Bitcoin besitzt zur Ausführung von Transaktionen auch eine Skript-Sprache. Diese ist im Gegensatz zu Ethereum bewusst eingeschränkt und nicht turingmächtig. Da man keine Schleifen programmieren kann, führt dies dazu, dass man Code mehrfach wiederholen muss. Dies führt dazu, dass Bitcoin-Transaktionen, die Smart Contracts ausdrücken, mehr Platz in der Blockchain einnehmen. Ethereum muss sich aufgrund der Turing-Vollständigkeit um das Halteproblem kümmern. Durch die Angabe eines maximalen Gas-Werts stellt Ethereum sicher, dass die Ausführung einer Transaktion spätestens nach einer gewissen Zeit abgebrochen wird.

Betrags-Blindheit

Der Betrag, der einer Bitcoin-Adresse zugeschrieben wird, kann entweder ganz oder gar nicht ausgegeben werden. Dies ist bei Ethereum nicht der Fall. Ethereum erlaubt es Transaktionen nur einen Bruchteil des Account Guthabens zu versenden.

Blockchain-Blindheit

In Bitcoin kann man bei der Ausführung von Transaktionen nicht auf Blockchain Daten wie beispielsweise vorherige Blockhash-Werte, Zeitstempel oder Nonce-Werte zugreifen. Eine Transaktion, die einen vorherigen Blockhash ausliest und als Zufallsquelle verwendet, ist mit der Bitcoin Skript-Sprache nicht möglich.

⁴Im resultierenden Systemzustand aus Abbildung 3.2 fehlt die an den Miner gezahlte Transaktionsgebühr von 1 Ether. Diese zahlt sich der Miner im letzten Schritt (siehe Abbildung 3.1) zusammen mit den restlichen Transaktionsgebühren des Blocks aus.

Fehlender Zustand

Die Ethereum Accounts ermöglichen es einen weitaus komplexeren Systemzustand abzubilden. Bei Bitcoin besteht der Systemzustand lediglich aus der Menge aller Adressen und der Anzahl Bitcoin, die diese besitzen.

3.2 Konzept

Der Ablauf des Spiels ist mit dem Ablauf aus Abschnitt 2.2 nahezu identisch. Die Unterschiede sind, dass das gesamte Spiel vom Nutzer initiiert wird und die Glücksspielanwendung ausschließlich den Spielstatus anzeigt. Dies führt dazu, dass die Spieler keinerlei Vertrauen in die Glücksspielanwendung haben müssen. Statt aufgrund leichter Überprüfbarkeit die letzte numerische Stelle des Blockhashs für die Gewinnerauswahl zu nutzen, kann bei Ethereum der gesamte Blockhash zur Gewinnerauswahl genutzt werden. Der Spieler kann aufgrund der Konsensregeln darauf vertrauen, dass das Ethereum Netzwerk die Ausführung des Contract Codes korrekt durchführt und die Modulo-Funktion zur Gewinnerauswahl korrekt berechnet. Die Nutzung des gesamten Blockhashs als Zufallsquelle ermöglicht Töpfe mit einer beliebiger Größe. Der Ablauf des Spiels lässt sich durch den folgenden Zustandsautomat beschreiben.

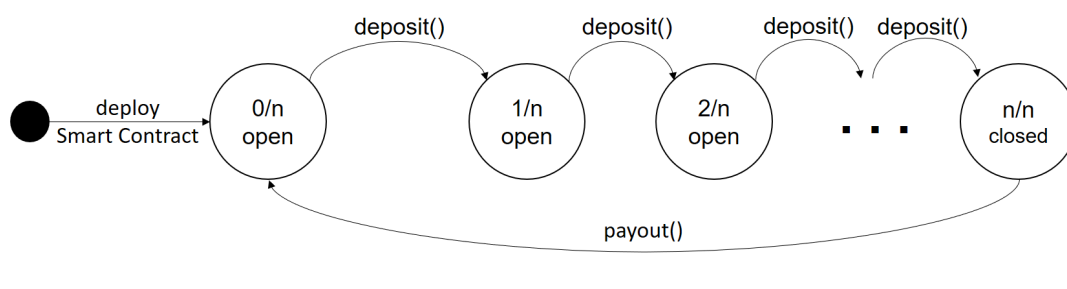


ABBILDUNG 3.3: Smart Contract Automat

Durch den Aufruf der `deploy` Funktion wird der Smart Contract in einer Transaktion an das Netzwerk gesendet und innerhalb eines Blocks in die Blockchain aufgenommen. Im Smart Contract sind der Einzahlungsbetrag und die Anzahl Teilnehmer n fest definiert. Ab diesem Moment kann der Code des Smart Contracts nicht mehr verändert werden. Im initialen Zustand ist der Topf leer, da noch kein Spieler eingezahlt hat. Nun können genau n Einzahlungen von Spielern durch den Aufruf der `deposit` Funktion getätigt werden. Dazu benötigen die Spieler lediglich einen Ethereum Client, ausreichend Ether⁵ und die Adresse des Smart Contracts. Ab dem Zeitpunkt an dem der Smart Contract die letzte Einzahlungstransaktion empfängt, wird der Topf geschlossen und die Blocknummer für die Gewinnerauswahl festgelegt. Der Block, der den Gewinner des Topfs entscheidet, muss in der Zukunft liegen und darf nicht vorher bekannt sein, da sonst Betrugsmöglichkeiten entstehen. Zum Zeitpunkt der letzten Einzahlungstransaktion ist es nicht möglich aus dem Smart Contract Code heraus auf den Blockhash des Blocks zuzugreifen in dem sich die letzte Einzahlungstransaktion befindet. Dies liegt daran, dass die Miner das Resultat der Zustandsveränderung aller Transaktionen des Blockes in den Blockheader schreiben müssen und erst anschließend den Blockhash berechnen. Die Transaktionen, die den Contract Code ausführen, können somit nicht auf den Blockhash zugreifen, da dieser zum Zeitpunkt der Code-Ausführung noch nicht feststeht. Es ist somit unumgänglich nach der letzten Einzahlungstransaktion eine Funktion aufzurufen, die den Gewinner auswählt, die Auszahlung startet und den Topf anschließend für ein neues Spiel wieder öffnet. Dies ist die in Abbildung 3.3 gezeigte `payout` Funktion.

⁵Einzahlungsbetrag plus Transaktionskosten.

3.3 Umsetzung

3.3.1 Überblick

Genau wie bei Bitcoin besteht die Möglichkeit die Glücksspielanwendung entweder direkt mithilfe eines *Light-Nodes* oder indirekt über einen *Full-Node* mit dem Ethereum Netzwerk kommunizieren zu lassen. Für den Ethereum-Teil dieser Masterarbeit findet die Kommunikation indirekt über einen Full-Node statt. Dies ist in Abbildung 3.4 verdeutlicht. Der Full-Node empfängt Transaktionen und Blöcke, validiert diese und aktualisiert kontinuierlich den Zustand der, durch die Transaktionen veränderten, Ethereum Accounts. Über die RPC Schnittstelle stellt er diese Daten nach außen bereit. Die Java Bibliothek Web3J [40] erleichtert den Aufruf der RPC Schnittstelle des Full Nodes. Anders als bei Bitcoin benötigt die Glücksspielanwendung keine eigene Datenbank, da der Zustand des aktuellen Topfs im Smart Contract und somit "in der Blockchain" gespeichert ist.

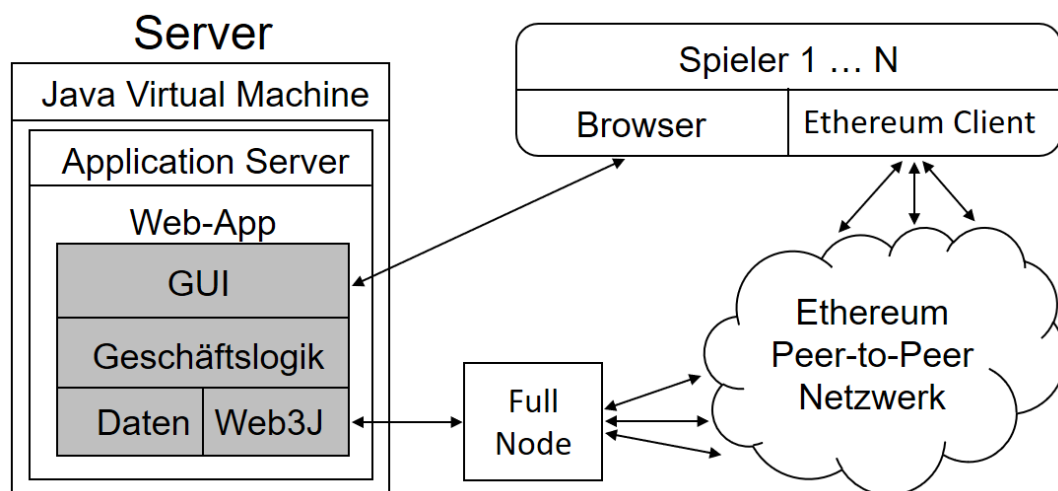


ABBILDUNG 3.4: Ethereum: Netzwerk Integration

Möchte man eine Anwendung direkt in das Ethereum Netzwerk integrieren, bietet sich die Java Bibliothek EthereumJ [14] an.

3.3.2 Smart Contract

Die folgenden Codestücke beschreiben den TrustlessGambling Smart Contract in der Programmiersprache Solidity. Dokumentation zur Solidity Programmiersprache findet man unter [17].

Datenmodell

Das folgende Codestück zeigt den Rahmen, alle Variablen und den Konstruktor des Smart Contracts.

```

1 pragma solidity ^0.4.0;
2 contract TrustlessGambling {
3     // constants
4     uint8 public constant NBR_OF_SLOTS =3;
5     uint public constant EXPECTED_POT_AMOUNT=1000;// WEI

```

```

6      uint8 public constant PAYOUT_BLOCK_OFFSET =1;
7      // pot values
8      uint public nbrOfParticipants;
9      address[NBR_OF_SLOTS] public depositAddresses;
10     address[NBR_OF_SLOTS] public payoutAddresses;
11     uint public closingBlockNumber;
12     uint public payoutBlockNumber;
13     bytes32 public payoutBlockHash;
14     uint public winner; // 0 -> NBR_OF_SLOTS-1
15     bool public potClosed;
16     uint public nbrOfMissedPayouts;
17     // constructor
18     function TrustlessGambling() public {
19         nbrOfParticipants = 0;
20         potClosed = false;
21         nbrOfMissedPayouts = 0;
22     }
23 }

```

Zeile 1 definiert in welcher Solidity-Version der Smart Contract geschrieben ist. Dies muss vom Compiler bei der Übersetzung zu Bytecode berücksichtigt werden. Zeile 2 legt den Namen des Smart Contracts fest. Über die beiden Konstanten in Zeile 4 und 5 kann man die Anzahl Spieler und den von jedem Spieler erwarteten Einzahlungsbetrag festlegen. Die in Zeile 6 definierte Konstante legt fest, welcher Block (ab der letzten Einzahlungstransaktion) den Gewinner bestimmt. Diese Werte können nach der Bereitstellung des Smart Contracts nicht mehr verändert werden. Die Variablen von Zeile 8 bis Zeile 16 werden vom Smart Contract manipuliert und speichern den Zustand des aktuellen Topfs. Zeile 8 speichert wie viele Teilnehmer bereits eingezahlt haben. Zeile 9 und 10 speichern die Ein- und Auszahlungsadressen⁶ der aktuellen Teilnehmer. Die Zeilen 11 bis 14 speichern alle für die Gewinnerauswahl benötigten Werte. Zeile 15 definiert über den Wahrheitswert `potClosed`, ob der Topf offen ist und Einzahlungen stattfinden können. Der Nutzen des Wertes aus Zeile 16 wird im Rahmen des **Auszahlungen** Abschnitts erklärt. Zeile 18 bis 22 beinhalten den einmalig bei der Bereitstellung des Smart Contracts aufgerufenen Konstruktor. Alle Variablen des Smart Contracts sind zur Schaffung maximaler Transparenz mit dem Schlüsselwort `public` markiert. Dies erlaubt es den Nutzern alle Werte des Smart Contract abzurufen.

Einzahlungen

Einzahlungen finden über die beiden `deposit` Methoden statt. Diese sind mit dem Schlüsselwort `payable` markiert. Dies bedeutet, dass Transaktionen einen Ether-Betrag beim Aufruf dieser Methoden angeben können.

```

1 function deposit() payable public {
2     deposit(msg.sender);
3 }
4 function deposit(address _payout) payable public {
5     assert(!potClosed);
6     assert(msg.value == EXPECTED_POT_AMOUNT);
7     depositAddresses[nbrOfParticipants] = msg.sender;
8     payoutAddresses[nbrOfParticipants] = _payout;

```

⁶Die Einzahlungsadressen werden nur zur Anzeige für die GUI-Anwendung abgespeichert und sind für das eigentliche Spiel irrelevant. Das Weglassen dieser Adressen würde zu günstigeren Transaktionskosten für Einzahlungstransaktionen führen.


```

9      nbrOfParticipants++;
10     if (nbrOfParticipants == NBR_OF_SLOTS) {
11         closingBlockNumber = block.number;
12         payoutBlockNumber = closingBlockNumber +
            PAYOUT_BLOCK_OFFSET;
13         potClosed = true;
14     }
15 }

```

Nutzt der Spieler die Methode aus Zeile 1 wird als Auszahlungsadresse einfach die Adresse der Transaktion verwendet. Nutzt der Spieler die Methode aus Zeile 4 hat er die Möglichkeit eine beliebige Auszahlungsadresse anzugeben. Bei der Einzahlung wird zunächst in Zeile 5 geprüft, ob der Topf offen ist. Ist dies der Fall prüft Zeile 6, dass der Transaktionsbetrag mit dem erwarteten Einzahlungsbetrag übereinstimmt. Anschließend wird die Ein- und Auszahlungsadresse abgespeichert und die aktuelle Anzahl Teilnehmer um eins erhöht. Zeile 10 prüft, ob es sich um die letzte Einzahlungstransaktion handelt und schließt gegebenenfalls den Topf. Vor dem Schließen des Topfs wird in Zeile 11 die aktuelle Blocknummer abgespeichert und anschließend die Blocknummer für die Gewinnerauswahl berechnet.

Auszahlungen

Auszahlungen finden durch den Aufruf der payout Methode statt. Diese ist nicht mit dem Schlüsselwort payable markiert und erwartet keinen Ether-Betrag beim Aufruf.

```

1 function payout() public {
2     assert(potClosed);
3     assert(block.number > payoutBlockNumber);
4     payoutBlockHash = block.blockhash(payoutBlockNumber);
5     if (payoutBlockHash == 0) {
6         nbrOfMissedPayouts++;
7     } else {
8         winner = uint256(payoutBlockHash) % NBR_OF_SLOTS;
9         address winnerAddress = payoutAddresses[winner];
10        uint amount = EXPECTED_POT_AMOUNT * NBR_OF_SLOTS;
11        amount +=
            EXPECTED_POT_AMOUNT * NBR_OF_SLOTS * nbrOfMissedPayouts;
12        winnerAddress.transfer(amount); // send pot amount to
            winner
13        nbrOfMissedPayouts = 0;
14    }
15    potClosed = false;
16    nbrOfParticipants = 0;
17 }

```

Die Methode kann nur aufgerufen werden, falls der Topf geschlossen ist und die aktuelle Blocknummer bereits höher als die Blocknummer für die Gewinnerauswahl ist. Sind diese Bedingungen erfüllt, hängt der weitere Verlauf der Abarbeitung der payout Methode vom Zeitpunkt des Methodenaufrufs ab. Smart Contracts können laut einer Konvention bei ihrer Ausführung nur auf die Werte der 256 letzten Blockheader zugreifen (siehe [18]). Wenn der in Zeile 4 angefragte payoutBlockHash älter als 256 Blöcke ist, gibt block.blockhash(<number>) den Wert 0 zurück und Fall 1 tritt ein.

1. Fall (Zeile 5): Der Aufruf der payout Methode findet zu spät statt. Es kommt zu keiner Auszahlung, da der Smart Contract nicht auf den entscheidenden

Blockhash zugreifen kann. Der Smart Contract erhöht die `nbrOfMissedPayouts` Variable um eins. Dies führt dazu, dass der Betrag des Topfs in den nächsten Topf verschoben wird.

2. Fall (Zeile 8): Der Aufruf der `payout` Methode findet rechtzeitig statt. Der Smart Contract berechnet den Gewinner indem er den Blockhash zu einem Integer castet und diese sehr hohe Zahl modulo der Anzahl Teilnehmer rechnet. Anschließend wird der korrekte Auszahlungsbetrag berechnet und in Zeile 12 an die Auszahlungsadresse des Gewinners versandt.

Zum Schluss wird der Topf wieder geöffnet (Zeile 15) und die Anzahl der teilnehmenden Spieler auf 0 gesetzt (Zeile 16).

3.3.3 Smart Contract Bereitstellung

Nachdem man den Smart Contract programmiert hat, muss man ihn zu Bytecode kompilieren und anschließend in einer Transaktion an das Ethereum Netzwerk senden. Der in Solidity geschriebene Contract Code kann mithilfe eines Online-Compilers ⁷ kompiliert werden. Das Kompilieren erzeugt die Dateien `TrustlessGambling.bin` und `TrustlessGambling.abi`. Diese enthalten den Bytecode und das Smart Contract Application Binary Interface. Um in der Programmiersprache Java mit dem Smart Contract interagieren zu können, stellt Web3J sogenannte Comandline Tools zur Verfügung. Durch den Aufruf des folgenden Befehl wird die Java Klasse `TrustlessGambling` generiert:

```
web3j solidity generate TrustlessGambling.bin TrustlessGambling.abi
-o /path/to/src/main/java -p com.ossel.gamble.ethereum.generated
```

Da zur Bereitstellung des Smart Contracts auch die entsprechende Transaktionsgebühr gezahlt werden muss, wird eine Wallet benötigt. Web3J hilft auch bei diesem Schritt. Der folgende Befehl leitet die Generierung einer Wallet ein:

```
web3j wallet create
```

Über die Kommandozeile muss der Benutzer den gewünschten Wallet-Dateinamen und ein Passwort angeben. In diesem Beispiel wird der Dateiname `ethereum.json` und das Passwort `changeit` verwendet. Anschließend wird die Wallet generiert und die Ethereum Account Adresse ausgegeben. Eine genaue Beschreibung der Web3J Comandline Tool Befehle findet man unter [38]. Bevor der Smart Contract durch eine Transaktion veröffentlicht wird, muss zunächst eins der folgenden Ethereum Netzwerke gewählt werden:

1. Mainnet: Genau wie bei Bitcoin handelt es sich bei diesem Netzwerk um das Hauptnetzwerk.
2. Ropsten Testnetz: Hierbei handelt es sich um ein Testnetz, das Proof-of-Work als Konsensalgorithmus verwendet. Es ist dem Ethereum Mainnet am ähnlichsten. Der auf diesem Netzwerk ausgetauschten Ether-Währung wird allerdings kein finanzieller Wert zugemessen.
3. Rinkeby Testnetz: Hierbei handelt es sich um ein Testnetz, das nicht Proof-of-Work sondern das Clique-Proof-of-Authority Protokoll als Konsens-Algorithmus verwendet. Im Gegensatz zu Proof-of-Work wird ein Konsens durch das Signieren von Blöcken durch bekannte Teilnehmer gewährleistet. Das Ethereum Improvement Proposal [9] beschreibt den verwendeten Vorgang detailliert.

⁷<http://remix.ethereum.org>

Das Problem bei der Verwendung des Proof-of-Work Algorithmus auf einem Testnetz ist, dass Miner für ihren Stromverbrauch nur in der wertlosen Testnetz-Währung bezahlt werden. Daraus resultiert, dass solch ein Testnetz nur eine sehr geringe Hashrate aufweist. Ein Angreifer, der eine signifikante Hashrate besitzt, kann alle neuen gültigen Blöcke des restlichen Netzwerks verwerfen und selber die längste Blockkette erzeugen. Erzeugt der Angreifer dabei nur noch Blöcke, die keine Transaktionen beinhalten, kann er dadurch das Testnetz für eine gewisse Zeit lahmlegen. Das Rinkeby Testnetz ist gegen solche Angriffe resistenter und daher im allgemeinen das stabilere Testnetzwerk.

Um Ether auf dem Rinkeby Testnetz zu erhalten, verwendet man eine sogenannte Faucete⁸. Dabei handelt es sich um eine Webseite der Testnetzbetreiber, die in regelmäßigen Abständen Kleinstbeträge an Ether verschenkt. Über die Eingabe einer Ethereum Account Adresse kann man auf diese Weise in den Besitz von Währungseinheiten kommen.

Nachdem die vorher erzeugte Wallet über Ether verfügt, kann man den Smart Contract mit Hilfe von Web3J bereitstellen. Abbildung 3.5 listet die dazu verwendeten Klassen und die generierte TrustlessGambling Klasse auf.

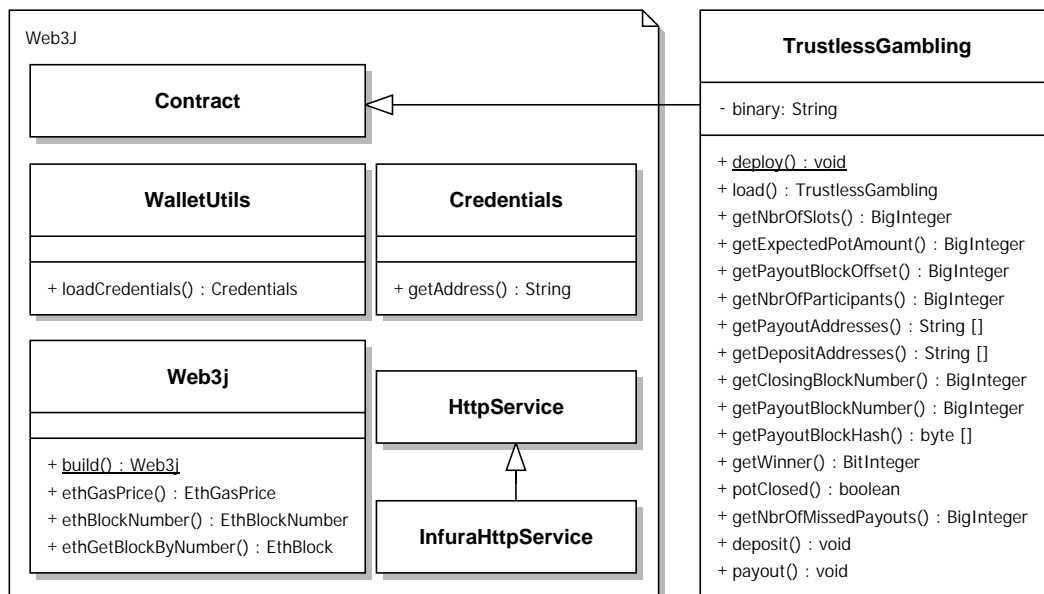


ABBILDUNG 3.5: Klassendiagramm Web3J

Der folgende Java Code sorgt dafür, dass der von Web3J angesprochene Full-Node den Smart Contract in einer Transaktion an das Rinkeby Testnetz sendet.

```

1 public void createContract() throws Exception {
2     String WALLET_FILENAME = "ethereum.json";
3     String WALLET_PASSWORD = "changeit";
4     long GAS_LIMIT = 1000000;
5     Web3j web3j = Web3j.build(new
        InfuraHttpService("https://rinkeby.infura.io/" +
        UserConfiguration.API_KEY));
6     BigInteger currentGasPrice =
        web3j.ethGasPrice().send().getGasPrice();
  
```

⁸<https://faucet.rinkeby.io/>

```

7   ClassLoader classLoader = getClass().getClassLoader();
8   File walletFile = new
      File(classLoader.getResource(WALLET_FILENAME).getFile());
9   Credentials credentials =
      WalletUtils.loadCredentials(WALLET_PASSWORD,
      walletFile.getAbsolutePath());
10  System.out.println("Account address = " +
      credentials.getAddress());
11  TrustlessGambling contract = TrustlessGambling.deploy(web3j,
      credentials, currentGasPrice,
      BigInteger.valueOf(GAS_LIMIT)).send();
12  String status =
      contract.getTransactionReceipt().get().getStatus();
13  if ("0x1".equals(status)) {
14      String address = contract.getContractAddress();
15      System.out.println("Contract address = " + address);
16      System.out.println("TXN hash = " +
          contract.getTransactionReceipt().get().getTransactionHash());
17      System.out.println("Gas used = " +
          contract.getTransactionReceipt().get().getGasUsed());
18  } else {
19      System.out.println("Smart contract could not be deployed.");
20  }
21 }

```

In Zeile 5 wird der Web3J Service erzeugt. Dieser kümmert sich um die Kommunikation mit dem Full-Node. Die übergebene URL legt die Adresse des Full-Nodes fest. Infura⁹ ist dabei ein Service, der sich auf das Hosting von Ethereum Full-Nodes spezialisiert hat. Mit Hilfe eines API Schlüssels kann man sich zu seinem Full-Node verbinden. Statt des von Infura betriebenen Nodes kann man auch einen eigens betriebenen Full-Node verwenden. Dies hat den Vorteil, dass man die volle Kontrolle behält. In diesem Fall verwendet man statt der `InfuraHttpService` Klasse direkt die Oberklasse `HttpService`. Zeile 6 fragt den Full-Node nach dem aktuell zu bezahlenden Gaspreis. Zeile 8 lädt die durch die Web3J Comandline Tools erzeugte Wallet. Mithilfe dieser werden unter Zuhilfenahme des Passworts die im nächsten Schritt verwendeten Credentials geladen. In Zeile 11 wird der Smart Contract durch den Aufruf der statischen `deploy` Methode in einer Transaktion an das Netzwerk gesendet. Dabei wird ein Gaslimit von einer Million WEI festgelegt. Die Ausführung des oben gezeigten Java Codes führt zu der folgenden Ausgabe:

```

Account address = 0x2201f3919589b519135ce977cc0906c9481069b2
Contract address = 0x25c3136145fbd7f3b9217e58e2fabe3eb1928705
TXN hash = 0x06dce3c460b4caa595c5cc0f81ac78e7c70eeb1e89d3e0ea88e60dbce1
Gas used = 825846

```

Das Gaslimit von einer Million WEI hat ausgereicht und der Smart Contract befindet sich nun in der Blockchain des Rinkeby Testnetzes. In einem Blockchain Explorer kann man die Details der Transaktion¹⁰ und den kompilierten Contract Code¹¹ anschauen.

⁹<https://infura.io/>

¹⁰<https://rinkeby.etherscan.io/tx/0x06dce3c460ac78e7c70eeb1e89d3e0ea88e60dbce1>

¹¹<https://rinkeby.etherscan.io/address/0x25c3136145fbd7f3b9217e58e2fabe3eb1928705>

3.3.4 Geschäftslogik Glücksspielanwendung

Die Glücksspielanwendung zeigt lediglich den aktuellen Zustand des Smart Contracts an. Die gesamte Geschäftslogik des Smart Contracts wird vom Ethereum Netzwerk ausgeführt. Sollte die Glücksspielanwendung aufgrund technischer Fehler ausfallen, hat dies keinerlei Auswirkung auf das eigentliche Spiel. Die Geschäftslogik der Glücksspielanwendung fragt lediglich in regelmäßigen Abständen beim Full-Node an, ob eine Änderung des Smart Contract Zustands stattgefunden hat. Abbildung 3.6 liefert einen ersten Überblick über die dazu verwendeten Klassen.

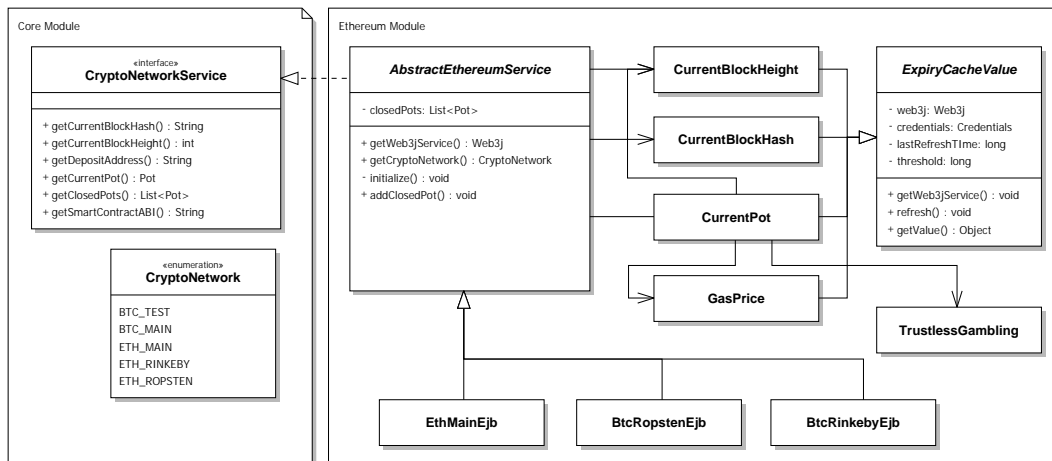


ABBILDUNG 3.6: Klassendiagramm Ethereum

Core Module

Das **CryptoNetworkService** Interface wurde um die Methode `getSmartContractABI` erweitert. Bei dem Smart Contract **Application Binary Interface** handelt es sich um die statische zur Compile-Zeit bestimmte Schnittstellenbeschreibung, die festlegt wie man mit dem Smart Contract interagieren kann. Das Smart Contract ABI wird üblicherweise im JSON Format angegeben.

Zu dem **CryptoNetwork** Enum sind nun die zusätzlichen Werte **ETH_MAIN**, **ETH_RINKEBY**, **ETH_ROPSTEN** hinzugekommen. Über diese Werte kann man steuern mit welchem Ethereum Netzwerk die Anwendung kommunizieren soll. Die Klassen **Pot** und **Participant** haben sich nicht verändert.

Ethereum Module: AbstractEthereumService

Die abstrakte Klasse **AbstractEthereumService** implementiert die **CryptoNetworkService** Schnittstelle. Die Klassen **EthMainEjb**, **EthRopstenEjb** und **EthRinkebyEjb** legen lediglich über den verwendeten **Web3j** Service fest, welches Netzwerk die Anwendung ansprechen soll. Die abstrakte Klasse **AbstractEthereumService** implementiert alle vom **CryptoNetworkService** Interface geforderten Methoden. Die Methoden `getDepositAddress` und `getSmartContractABI` geben lediglich die statische Smart Contract Adresse und den Smart Contract ABI JSON String zurück. Die Methoden `getCurrentBlockHash`, `getCurrentBlockHeight` und `getCurrentPot` geben gecachte Werte des jeweiligen **ExpiryCacheValue** durch den Aufruf der `getValue` Methode an die Webanwendung zurück. Die Klassen **CurrentBlockHeight**, **CurrentBlockHash**, **GasPrice** und **CurrentPot** erweitern die abstrakte **ExpiryCacheValue**

Klasse. Diese sorgt dafür, dass die Werte erst nach dem Ablauf einer gewissen konfigurierbaren Zeit (threshold) automatisch neu beim Full-Node angefragt werden. Dies verhindert, dass der Full-Node durch zu viele Anfragen überlastet wird. Alle Cache-Werte werden in der initialize Methode der AbstractEthereumService Klasse initialisiert.

```

1  @PostConstruct
2  private void initialize() {
3      log.info("#### start " + getClass().getSimpleName() + " network
         service ####");
4      blockHeightCache = new CurrentBlockHeight(getWeb3jService(),
         getCredentials());
5      log.info("blockHeight=" + blockHeightCache.getValue().intValue());
6      blockHashCache = new CurrentBlockHash(getWeb3jService(),
         getCredentials(), blockHeightCache);
7      log.info("blockHash=" + blockHashCache.getValue());
8      gasPriceCache = new GasPrice(getWeb3jService(),
         getCredentials());
9      log.info("gasPrice=" + gasPriceCache.getValue().intValue());
10     currentPotCache = new CurrentPot(getWeb3jService(),
         getCredentials(), gasPriceCache, blockHeightCache,
11         UserConfiguration.CONTRACT_ADDRESS);
12     log.info("currentPot=" + currentPotCache.getValue().toString());
13 }

```

Ethereum Module: CurrentBlockHash

Der folgende Code zeigt beispielhaft die Implementierung des CurrentBlockHash ExpieryCacheValue.

```

1  public class CurrentBlockHash extends ExpiryCacheValue {
2
3      CurrentBlockHeight blockHeight;
4
5      public CurrentBlockHash(Web3j web3j, Credentials credentials,
         CurrentBlockHeight blockHeight) {
6          super(web3j, credentials, 5 * SECOND);
7          this.blockHeight = blockHeight;
8      }
9
10     /**
11      * automatically refresh if cache value expired
12      */
13     @Override
14     protected void refresh() {
15         String hash = "error";
16         DefaultBlockParameterNumber number = new
         DefaultBlockParameterNumber(blockHeight.getValue());
17         try {
18             EthBlock ethBlock =
                 getWeb3jService().ethGetBlockByNumber(number,
                     false).send();
19             hash = ethBlock.getBlock().getHash();
20         } catch (IOException e) {
21             e.printStackTrace();
22         }
23         setValue(blockHash);

```

```

24     }
25 }

```

In Zeile 6 wird konfiguriert, dass der aktuelle Blockhash maximal alle 5 Sekunden vom Full Node abgefragt wird. In Zeile 18 wird der neuste Block durch den Aufruf der `ethGetBlockByNumber` Methode angefragt. Über den Wahrheitswert-Parameter kann man entweder die gesamten Blockdaten oder nur die Header-Informationen beim Full-Node anfragen.

Ethereum Module: CurrentBlockHeight

Die Implementierung des `CurrentBlockHeight ExpieryCacheValues` greift auf die folgenden Zeilen Code zurück. Es findet maximal alle 5 Sekunden eine Anfrage an den Full-Node statt.

```

1 EthBlockNumber ethBlockNumber =
    getWeb3jService().ethBlockNumber().send();
2 BigInteger currentBlockNumber = ethBlockNumber.getBlockNumber();

```

Ethereum Module: GasPrice

Die Implementierung des `GasPrice ExpieryCacheValues` greift auf die folgenden Zeilen Code zurück. Es findet maximal alle 60 Sekunden eine Anfrage an den Full-Node statt.

```

1 EthGasPrice ethGasPrice = getWeb3jService().ethGasPrice().send();
2 BigInteger gasPrice = ethGasPrice.getGasPrice();

```

Ethereum Module: CurrentPot

Der `CurrentPot ExpieryCacheValue` verwendet die Klasse `TrustlessGambling` um den Zustand des Smart Contracts zu erfassen und durch die Klasse `Pot` abzubilden. Im Konstruktor des `CurrentPot ExpieryCacheValues` wird die Methode `createEmptyPot` aufgerufen.

```

1 private Pot createEmptyPot() throws Exception{
2     TrustlessGambling contract =
        TrustlessGambling.load(contractAddress, getWeb3jService(),
3         getCredentials(), gasPrice.getValue(),
            BigInteger.valueOf(5300000));
4     int nbrOfSlots = contract.NBR_OF_SLOTS().send().intValue();
5     long amount =
        contract.EXPECTED_POT_AMOUNT().send().longValue();
6     return new Pot(nbrOfSlots, amount);
7 }

```

Diese Methode fragt den Full-Node, wie viele Spieler und welcher Einzahlungsbetrag vom Smart Contract erwartet wird und erzeugt anschließend einen neuen leeren Topf. Der Zustand des Topfs wird durch den folgenden Code jedes Mal aktualisiert, wenn die `refresh` Methode des `ExpieryCacheValues` aufgerufen wird. Dies findet alle 10 Sekunden statt.

```

1 TrustlessGambling contract =
    TrustlessGambling.load(contractAddress, getWeb3jService(),
2     getCredentials(),
        gasPrice.getValue(), BigInteger.valueOf(5300000));
3 Pot pot = (Pot) this.value;

```

```

4  int potParticipants = pot.getNbrOfParticipants();
5  int actualParaticipants =
    contract.nbrOfParticipants().send().intValue();
6  if (actualParaticipants >= potParticipants) {
7      for (int i = potParticipants; i < actualParaticipants; i++) {
8          String depositAddress = getDepositAddress(contract, i);
9          String payoutAddress = getPayoutAddress(contract, i);
10         pot.addParticipant(new Participant(depositAddress,
            payoutAddress));
11     }
12 } else {
13     // pot has been reopened
14     int winner = contract.winner().send().intValue();
15     byte[] hashBytes = contract.payoutBlockHash().send();
16     String payoutBlockhash =
        javax.xml.bind.DatatypeConverter.printHexBinary(hashBytes);
17     if (new BigInteger(payoutBlockhash).intValue() == 0) {
18         pot.setState(
19             "Pot closed. Payout() too late. The amount has
                been added to the next pot.");
20     } else {
21         Block block = new Block("0x" +
            payoutBlockhash.toLowerCase(), winner);
22         pot.setPayoutBlock(block);
23         pot.setWinner(winner);
24         pot.setState("Pot closed. Winner is " +
            pot.getWinner().getPayoutAddress());
25     }
26     this.ethereumService.addClosedPot(pot);
27     this.value = createEmptyPot();
28 }
29
30 boolean potClosed = contract.potClosed().send();
31 if (potClosed) {
32     int closingBlockNumber =
        contract.closingBlockNumber().send().intValue();
33     int payoutBlockNumber =
        contract.payoutBlockNumber().send().intValue();
34     pot.setClosingBlockHeight(closingBlockNumber);
35     pot.setPayoutBlockHeight(payoutBlockNumber);
36     int currentBlockNumber =
        currentBlockHeight.getValue().intValue();
37     if (pot.getPayoutBlockHeight() > currentBlockNumber) {
38         pot.setState("Pot closed. Waiting for payout block.");
39     } else {
40         int diff = currentBlockNumber - pot.getPayoutBlockHeight();
41         int blocksLeft = (256 - diff); // solidity restriction
42         if (blocksLeft > 0) {
43             pot.setState("Pot closed. Call payout() during the
                next " + blocksLeft
44                 + " blocks. Otherwise the whole amount will be
                added to the next pot.");
45         } else {
46             pot.setState(
47                 "Pot closed. Payout() too late. The amount has
                been added to the next pot. Call payout()
                to open a new pot.");

```



```
48     }  
49     }  
50 }
```

Der Code unterscheidet zwischen der Anzahl Teilnehmer, die die Glücksspielanwendung lokal zwischenspeichert (Zeile 4) und der Anzahl Teilnehmer des Datenfeldes des Smart Contracts (Zeile 5). Wenn neue Teilnehmer durch den Aufruf der `deposit` Methode in den Smart Contract einzahlen, wird der Topf durch die Zeilen 7 bis 11 aktualisiert. Die Ein- und Auszahlungsadressen der neuen Teilnehmer werden dazu aus den Smart Contract Daten geladen. Durch die letzte Einzahlung wechselt der Smart Contract in den Status `closed`. Ab diesem Moment wird der Topf durch die Abarbeitung des Codes ab Zeile 30 aktualisiert. Zunächst werden die finalen `closingBlockNumber` und `payoutBlockNumber` Werte aus den Smart Contract Daten geladen und die `currentBlockNumber` durch den Full-Node bestimmt. Anschließend wird zwischen 3 Fällen unterschieden:

1. Zeile 38: Der zur Gewinnerauswahl benötigte Block wurde noch nicht gefunden. Dies bedeutet, dass ein Aufruf der `payout` Methode noch nicht möglich ist.
2. Zeile 43: Der zur Gewinnerauswahl benötigte Block wurde gefunden und die `payout` Methode kann aufgerufen werden. In diesem Fall wird dem Benutzer angezeigt, wie viel Zeit er noch für den Aufruf der `payout` Methode hat.
3. Zeile 46: Der zur Gewinnerauswahl benötigte Block wurde zwar gefunden, es sind allerdings bereits mehr als 256 Blöcke zwischen der letzten Einzahlung und dem aktuellen Zeitpunkt vergangen. Der Smart Contract wird bei dem Aufruf der `payout` Methode nicht auf den Blockhash für die Gewinnerauswahl zugreifen können. Der Gewinn wird zum nächsten Topf hinzugefügt.

Durch den Aufruf der `payout` Methode wird das Datenfeld, das die Anzahl der aktuellen Teilnehmer des Smart Contracts speichert, auf den Wert `Null` gesetzt. Da der lokal gespeicherte Topf aber noch alle Teilnehmer enthält, hat dies zur Folge, dass beim erneuten Aufruf der `refresh` Methode des `CurrentPot` `ExpiryCacheValue` die Bedingung aus Zeile 6 verletzt ist und der Code von Zeile 13 bis 27 ausgeführt wird. Diese Codezeilen laden den Gewinner und den `payoutBlockHash` aus den Daten des Smart Contracts. Wenn der `payoutBlockHash` den Wert `Null` hat, wurde die `payout` Methode zu spät aufgerufen. Nur in diesem Fall gibt es keinen Gewinner. Anschließend wird der aktuelle Topf zur Liste der abgearbeiteten Töpfe hinzugefügt und ein neuer Topf durch den Aufruf der `createEmptyPot` Methode erzeugt.

3.3.5 Grafische Benutzeroberfläche

Das folgende Beispiel betrachtet einen frisch auf dem Ethereum Rinkeby Testnetzwerk bereitgestellten Smart Contract.

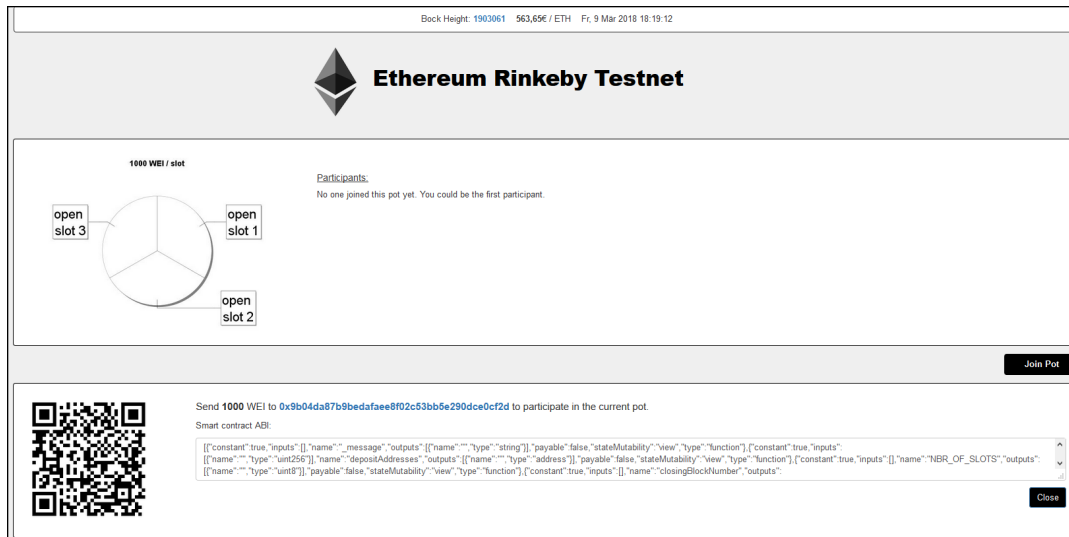


ABBILDUNG 3.7: Leerer Topf

Abbildung 3.7 zeigt einen Topf mit 3 freien Plätzen. Um dem Spiel beizutreten, muss der Spieler den Betrag von 1000 WEI an den Smart Contract senden. Genau wie bei Bitcoin wird dem Nutzer ein QR-Code angezeigt. Das Ethereum Improvement Proposal Nummer 681 [13] legt die Kodierung der Daten fest. Folgende Daten sind im QR Code enthalten:

“ethereum:0x9b04da87b9bedafaeef02c53bb5e290dce0cf2d/deposit?value=1000”. In diesem Beispiel verwenden wir für die Interaktion mit dem Netzwerk keinen Smartphone Client sondern die Webanwendung namens “My Ether Wallet”¹². Diese benötigt für die Interaktion mit dem Smart Contract sowohl die Contract Adresse als auch das Application Binary Interface.

¹²<https://www.myetherwallet.com/#contracts>

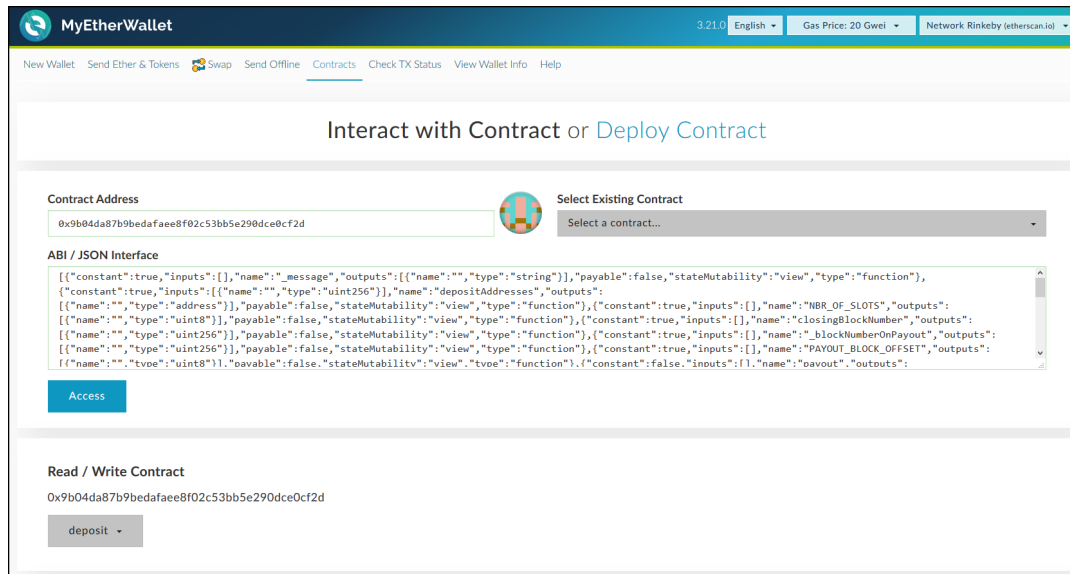


ABBILDUNG 3.8: My Ether Wallet

Nachdem der Nutzer das JSON-ABI wie in Abbildung 3.8 eingegeben hat, kann er über eine Dropdown-Liste die gewünschte Funktion des Smart Contracts aufrufen.

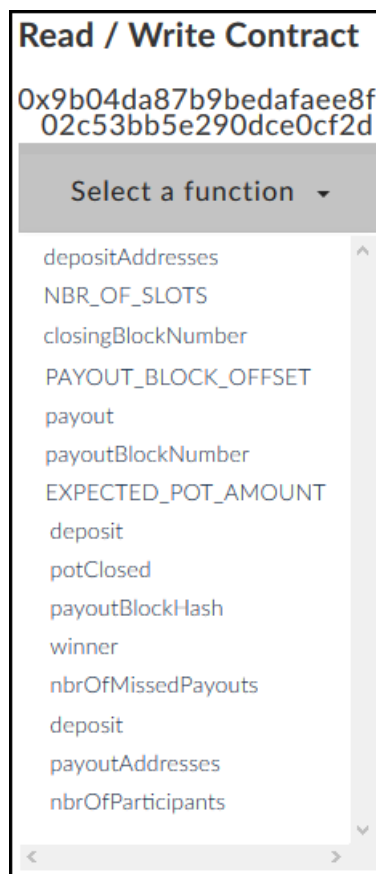


ABBILDUNG 3.9: Liste aller Smart Contract Funktionen

Abbildung 3.10 zeigt den Aufruf der Funktion, die die Höhe des vom Spieler erwarteten Geldbetrags zurückgibt.

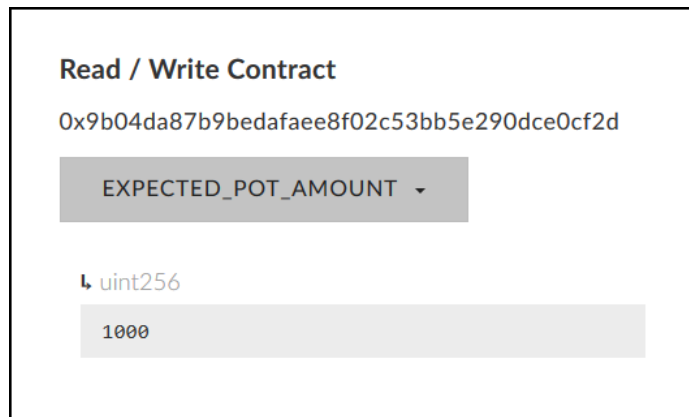


ABBILDUNG 3.10: Aufruf der EXPECTED_POT_AMOUNT Funktion

Da es sich lediglich um einen lesenden Zugriff handelt, wird keine Transaktion ans Netzwerk gesendet, beziehungsweise in die Blockchain geschrieben. Es fallen somit keine Transaktionskosten an.

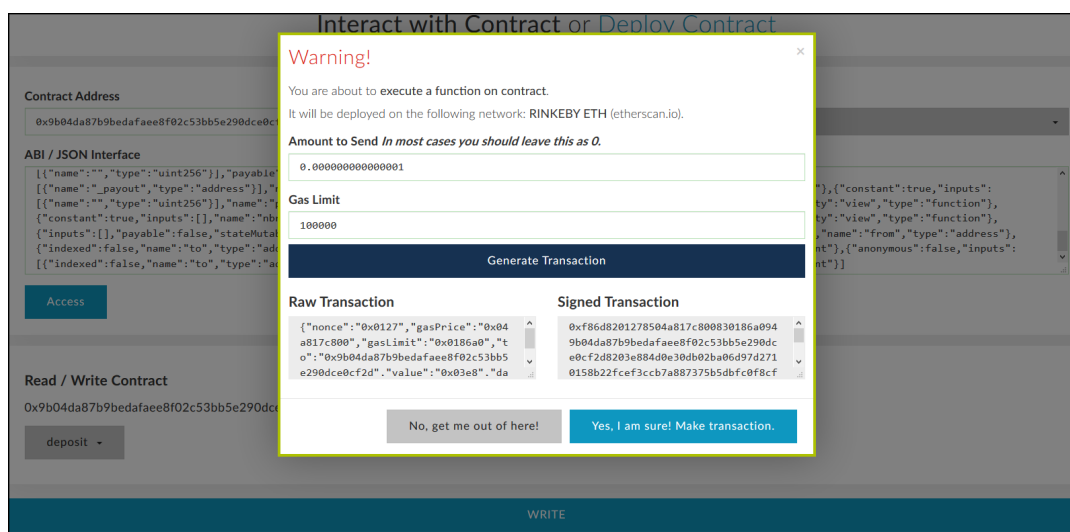


ABBILDUNG 3.11: Aufruf der deposit Funktion

Da der Nutzer nun nachgeprüft hat, dass der Smart Contract wirklich Zahlungen von 1000 WEI erwartet, kann er die deposit Funktion mit diesem Betrag aufrufen. Die Webseite erwartet den Betrag in der Einheit Ether. Die geforderten 1000 WEI entsprechen 0.0000000000000001 Ether. Die Umrechnung kann der Spieler mittels eines Online-Konverters¹³ durchführen. Nun muss die erstellte Transaktion nur noch signiert werden. Der Nutzer kann der Webseite dazu seinen privaten Schlüssel mitteilen oder die Signierung eigenständig durch ein sogenanntes Hardware Wallet durchführen. Die Herausgabe seines privaten Schlüssels an eine Webseite ist aus sicherheitstechnischer Sicht keine gute Praktik. Sollte der Webseitenbetreiber böse Absichten haben oder die Webseite gehackt werden, führt dies zum Verlust des durch

¹³<https://etherconverter.online/>

den Schlüssel kontrollierten Geldes. Eine sichere Variante ist die Verwendung eines Hardware Wallets. Dieses speichert alle privaten Schlüssel und führt die Signatur eigenständig durch. Der verwendete private Schlüssel verlässt somit niemals das Gerät.

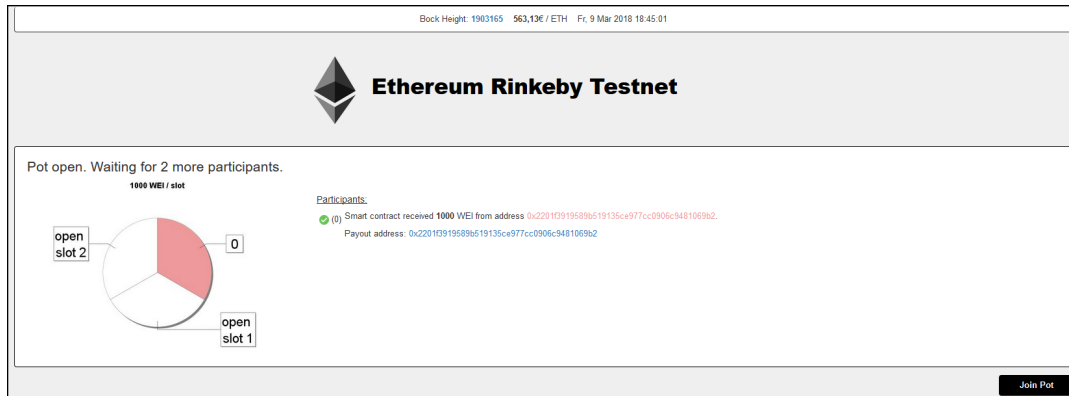


ABBILDUNG 3.12: Eingang der ersten Zahlung

Abbildung 3.12 visualisiert den Zustand des Smart Contracts nachdem die erste Einzahlungstransaktion in die Blockchain aufgenommen wurde.

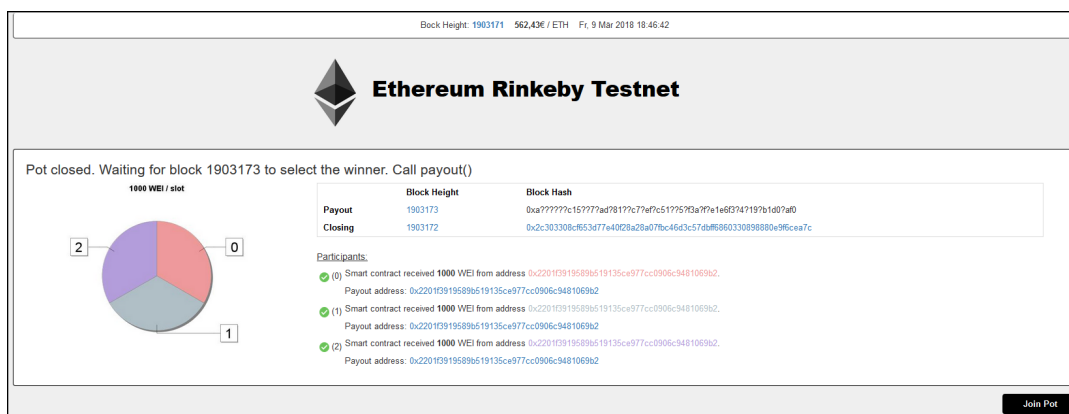


ABBILDUNG 3.13: Topf geschlossen

Abbildung 3.13 visualisiert den Zustand des Smart Contracts nachdem die letzte Einzahlungstransaktion in die Blockchain aufgenommen wurde. Der Smart Contract hat den Topf geschlossen und wartet nun, dass einer der Spieler die payout Funktion aufruft.

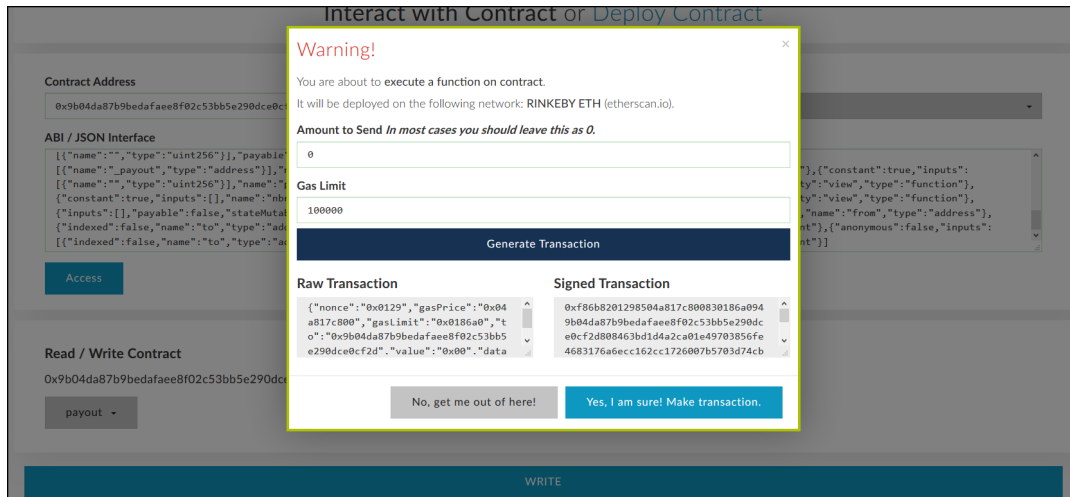


ABBILDUNG 3.14: Aufruf der payout Funktion

In Abbildung 3.14 ist gezeigt wie ein Spieler die payout Funktion aufruft. Durch den Aufruf dieser Funktion wird der Gewinner ausgewählt, die Auszahlung getätigt und der Topf wieder geöffnet.

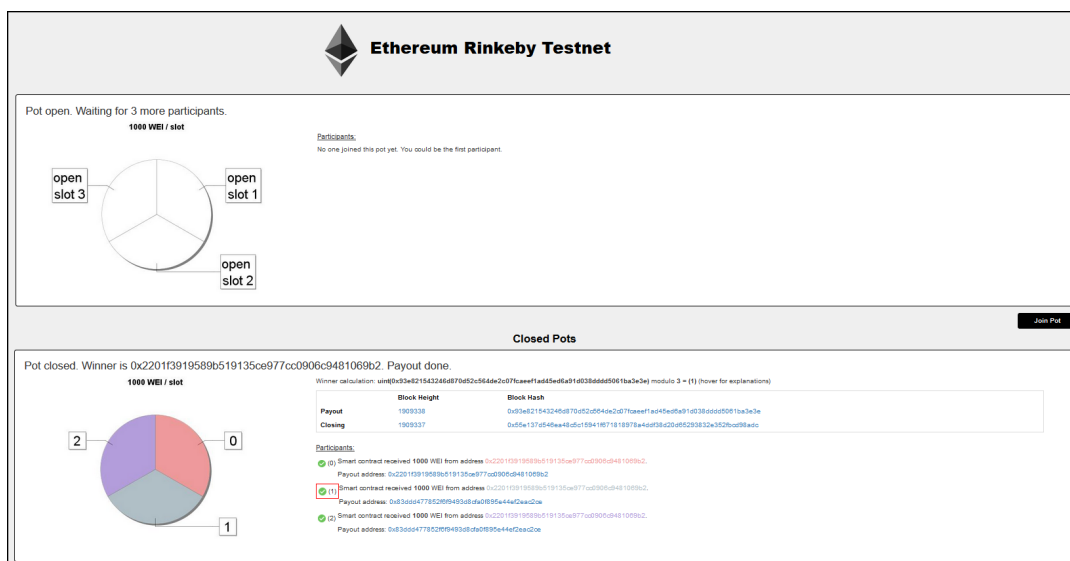
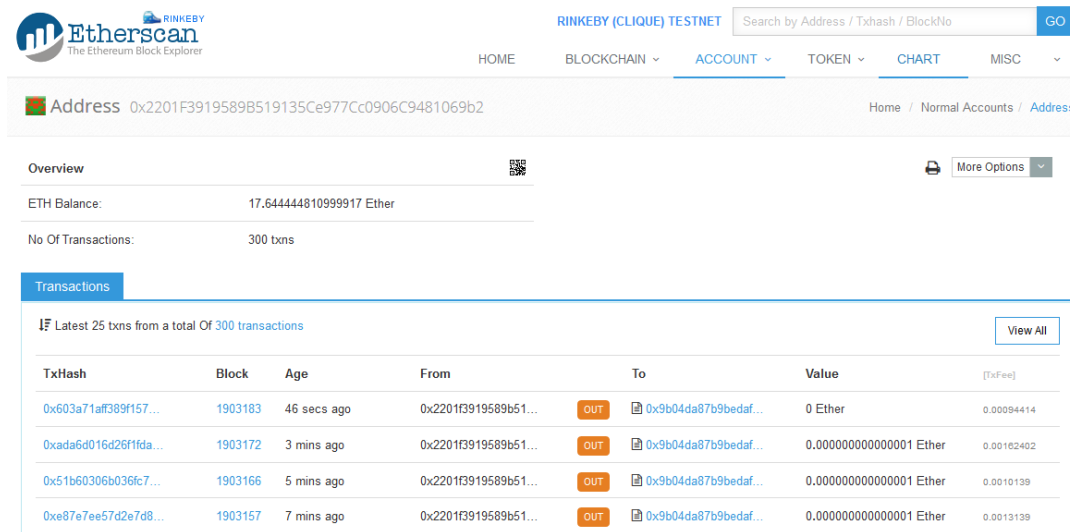


ABBILDUNG 3.15: Gewinner ausgewählt

Abbildung 3.15 zeigt den Gewinner des alten Topfs und den neu geöffneten Topf.



Etherscan RINKEBY
The Ethereum Block Explorer

RINKEBY (CLIQUE) TESTNET Search by Address / Txhash / BlockNo GO

HOME BLOCKCHAIN ACCOUNT TOKEN CHART MISC

Address 0x2201f3919589b519135ce977Cc0906C9481069b2 Home / Normal Accounts / Address

Overview

ETH Balance: 17.644444810999917 Ether

No Of Transactions: 300 txns

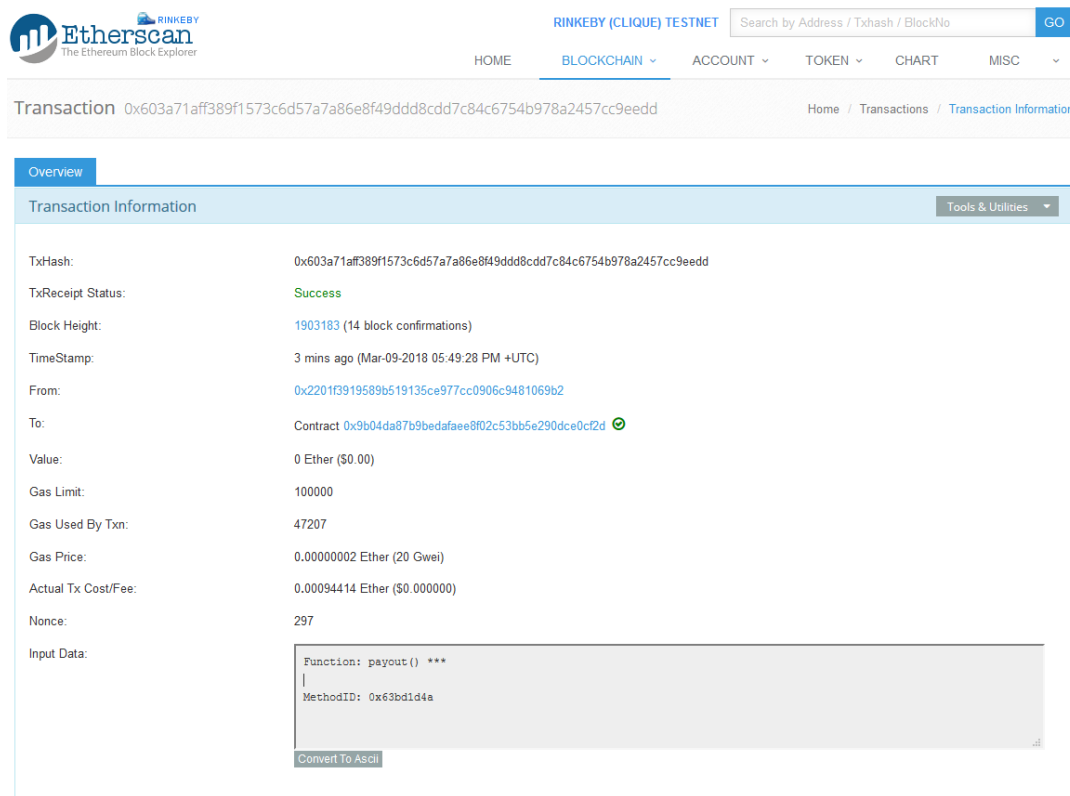
Transactions

Latest 25 txns from a total Of 300 transactions View All

TxHash	Block	Age	From	To	Value	[TxFee]
0x603a71aff389f157...	1903183	46 secs ago	0x2201f3919589b51...	OUT 0x9b04da87b9bedaf...	0 Ether	0.00094414
0xada6d016d26f1da...	1903172	3 mins ago	0x2201f3919589b51...	OUT 0x9b04da87b9bedaf...	0.0000000000000001 Ether	0.00162402
0x51b60306b036c7...	1903166	5 mins ago	0x2201f3919589b51...	OUT 0x9b04da87b9bedaf...	0.0000000000000001 Ether	0.0010139
0xe87e7ee57d2e7d8...	1903157	7 mins ago	0x2201f3919589b51...	OUT 0x9b04da87b9bedaf...	0.0000000000000001 Ether	0.0013139

ABBILDUNG 3.16: Block Explorer: Smart Contract

Abbildung 3.16 zeigt die 3 Einzahlungstransaktionen und die Transaktion, die die payout Funktion aufruft.



Etherscan RINKEBY
The Ethereum Block Explorer

RINKEBY (CLIQUE) TESTNET Search by Address / Txhash / BlockNo GO

HOME BLOCKCHAIN ACCOUNT TOKEN CHART MISC

Transaction 0x603a71aff389f1573c6d57a7a86e8f49ddd8cdd7c84c6754b978a2457cc9eedd Home / Transactions / Transaction Information

Overview

Transaction Information Tools & Utilities

TxHash: 0x603a71aff389f1573c6d57a7a86e8f49ddd8cdd7c84c6754b978a2457cc9eedd

TxReceipt Status: Success

Block Height: 1903183 (14 block confirmations)

TimeStamp: 3 mins ago (Mar-09-2018 05:49:28 PM +UTC)

From: 0x2201f3919589b519135ce977Cc0906C9481069b2

To: Contract 0x9b04da87b9bedafae8f02c53bb5e290dce0cf2d

Value: 0 Ether (\$0.00)

Gas Limit: 100000

Gas Used By Txn: 47207

Gas Price: 0.00000002 Ether (20 Gwei)

Actual Tx Cost/Fee: 0.00094414 Ether (\$0.000000)

Nonce: 297

Input Data:

```
Function: payout() ***
|
MethodID: 0x63bd1d4a
```

Convert To ASCII

ABBILDUNG 3.17: Block Explorer: Payout Transaktion

Abbildung 3.17 zeigt die Details der Transaktion, die die payout Funktion aufruft.

3.4 Evaluation

3.4.1 Prüfung der Anforderungen

Dieser Abschnitt behandelt in wieweit das beschriebene Konzept die in Abschnitt 1.4 aufgelisteten Anforderungen erfüllt. Die jeweilige Anforderung wird zunächst wiederholt und anschließend genauer untersucht.

1) Transparente Einzahlungen

Die Einzahlung jedes Endnutzers ist für jeden anderen Endnutzer nachprüfbar.

Einzahlungen geschehen genau wie bei Bitcoin innerhalb von Transaktionen, die in die Blockchain geschrieben werden. Diese Anforderung ist also auch im Falle von Ethereum erfüllt.

2) Gewinnerauswahl durch Zufallsfaktor

Die Auswahl des Gewinners ist von einem zufälligen Faktor abhängig, auf den weder die Anwendung noch die Endnutzer einen Einfluss haben.

Genau wie bei Bitcoin findet die Gewinnerauswahl basierend auf einem aus dem Proof-of-Work Blockhash statt. Die in Abschnitt 2.4.1 betrachtete Analyse gilt somit genauso für Ethereum außer, dass der Mining Reward 3 Ether beträgt und durchschnittlich alle 12 Sekunden ausgeschüttet wird. In Zukunft plant Ethereum von einem Proof-of-Work Algorithmus auf einen Proof-of-Stake Algorithmus umzusteigen. Proof-of-Stake und die daraus resultierenden Auswirkungen werden in Kapitel 4.2 betrachtet.

3) Nachprüfbarkeit des Zufallsfaktors

Jeder Endnutzer kann die Echtheit des zufälligen Faktors eigenständig nachprüfen.

Der zur Gewinnerauswahl verwendete Blockhash ist zum Zeitpunkt der Auszahlung bereits in der öffentlichen Blockchain verankert und kann somit überprüft werden.

4) Transparente Auszahlungen

Die Auszahlung an den Gewinner muss transparent und somit für jeden Endnutzer nachprüfbar sein.

Die Auszahlung wird durch die vom Nutzer initiierte payout Transaktion ausgelöst und vom Smart Contract vorgenommen. Das Verfahren der Auszahlung ist durch den unveränderlichen Smart Contract Code in Stein gemeißelt. Die Konsensregeln und die dahinter liegende Spieltheorie garantieren, dass dieser auch genauso ausgeführt wird. Obwohl die payout Transaktion nicht direkt Geld auf die Auszahlungsadresse des Gewinners überweist, kann der Endnutzer dennoch sicher sein, dass eine Auszahlung stattgefunden hat, wenn die payout Transaktion wie in Abbildung 3.17 im Status success vorliegt.

5) Fairheit des Spiels

Jeder Endnutzer besitzt die gleiche Gewinnwahrscheinlichkeit und niemand wird benachteiligt.

Damit keiner der Spieler einen Vorteil hat, muss jeder Topf-Platz die gleiche Gewinnwahrscheinlichkeit haben. Dies ist gegeben, falls jeder Teilnehmer a) die gleiche Anzahl Gewinnzahlen zugeordnet bekommt und b) die möglichen Blockhash-Werte für die Gewinnerauswahl gleichverteilt sind.

a) Statt wie bei Bitcoin ausschließlich die letzte Ziffer des Blockhashs für die Gewinnerauswahl zu verwenden und als Konsequenz lediglich Töpfe der Größe 2, 5 und 10 anzubieten, wird bei Ethereum der gesamte Blockhash zur Gewinnerauswahl verwendet. Dies führt zu beliebig großen Töpfen, bei denen einige Teilnehmer genau eine Gewinnzahl mehr haben können. Da jeder Spieler in der Praxis mehrere Millionen Gewinnzahlen hat, kann man diesen theoretischen Vorteil vernachlässigen.

b) Abschnitt 3.4.3 zeigt, dass die von Ethereum eingesetzte Keccak-256 Hashfunktion gleichverteilte Werte liefert.

3.4.2 Aufruf der Auszahlungstransaktion

Wie bereits in Abschnitt 3.2 betrachtet, ist der Aufruf einer Funktion zur Auszahlung unumgänglich. Da der Smart Contract dies nicht selber kann, muss der Aufruf entweder von außerhalb oder von einem anderen Smart Contract kommen.

Aufruf von außerhalb:

Der Aufruf kann wie in der Implementierung vom Gewinner ausgeführt werden. In diesem Fall zahlt der Gewinner die Transaktionsgebühr und erhält den gesamten Topf-Betrag. Der Gewinner ist dafür zuständig die Funktion rechtzeitig aufzurufen, da der Gewinn sonst in den nächsten Topf übergeht. Eine andere Möglichkeit ist es, dass die Glücksspielanwendung den Smart Contract überwacht und die *pay-out* Funktion rechtzeitig aufruft. In diesem Fall müsste die Transaktionsgebühr von der Glücksspielanwendung gezahlt werden oder Funktionalität in den Smart Contract eingebaut werden, die die Transaktionskosten vom Topf-Betrag abzieht und der Glücksspielanwendung zurückerstattet. Allerdings verlässt sich der Gewinner dann auf die Anwendung und geht dadurch ein Risiko ein.

Aufruf durch Smart Contract:

Man kann in der Theorie den Ansatz des Ethereum Alarm Clock Contracts¹⁴ verwenden, um eine gewünschte Smart Contract Funktion zu einem späteren Zeitpunkt auszuführen. Man spezifiziert dazu welche Funktion man wann (in welchem Blockzeitraum) ausführen möchte und zahlt für die anfallenden Transaktionsgebühren im Voraus. Dies erlaubt, dass sich eine ganze Reihe von Funktionen bei dem Alarm Clock Contract registrieren. Wird nun der Alarm Clock Contract von einem durch einen privaten Schlüssel kontrollierten Account ausgelöst, werden alle registrierten Funktionen aufgerufen. Leider liefert diese Vorgehensweise keine Garantie, da eine registrierte Funktion nur aufgerufen wird, falls der Alarm Clock Contract aufgerufen wird. Die Glücksspielanwendung müsste also einspringen, sobald niemand anderes bereit ist, den Alarm Clock Contract anzustoßen. Es handelt sich also lediglich

¹⁴<https://etherscan.io/address/0x6c8f2a135f6ed072de4503bd7c4999a1a17f824b>

um eine Vorgehensweise um Transaktionsgebühren mit anderen Ethereum Nutzern zu teilen. Weitere Informationen zum Ethereum Alarm Clock Contract findet man unter [27].

3.4.3 Verteilung der Hashfunktion Keccak-256

Ethereum verwendet die kryptographische Hashfunktion Keccak-256. Die folgende Monte-Carlo-Simulation legt nahe, dass die Hashwerte der Hashfunktion Keccak-256 gleichverteilt sind.

```
h=Keccak-256 n=1000000
for i 1 -> n
    hash = h(i);
    result[uint(hash)%10]++
```

Ausgabe:

```
result[0] = 99227
result[1] = 100479
result[2] = 100163
result[3] = 99804
result[4] = 99945
result[5] = 100208
result[6] = 100403
result[7] = 100438
result[8] = 100035
result[9] = 99298
```

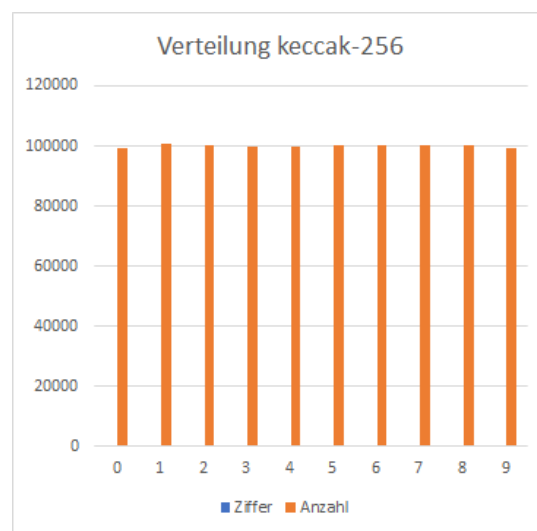


ABBILDUNG 3.18:
Verteilung der Keccak-
256 Hashfunktion

3.4.4 Sicherheit von Smart Contracts

Bei Smart Contracts handelt es sich um öffentliche, für jeden ausführbare und unveränderliche Software. Beinhaltet diese einen Software-Fehler, ist dieser ausnutzbar und kann nicht behoben werden. Smart Contracts verwalten in der Regel Geld oder Token, die einen finanziellen Wert repräsentieren. Bei der Entwicklung eines Smart Contracts ist somit oberste Vorsicht geboten. Das Beispiel des Smart Contracts *the-DAO*¹⁵ zeigt zu welchen fatalen Folgen eine Sicherheitslücke führen kann. Durch das Ausnutzen eines nicht trivialen Fehlers im Smart Contract Code schaffte es ein Hacker 3,6 Millionen Ether an eine von ihm kontrollierte Adresse auszuzahlen. Eine genaue Beschreibung des Angriffes findet man unter [11].

¹⁵The DAO ist ein, als Smart Contract realisierter, Kapitalfond, der es sich zur Aufgabe gemacht hat in Blockchain Technologie zu investieren. Im April 2016 haben über elftausend Investoren Kapital von mehr als 150 Millionen Dollar in den Fond eingezahlt. Als Gegenleistung haben die Investoren eine entsprechende Anzahl Token als eine Art Stimmrecht erhalten. Investitionsentscheidungen werden von den Investoren durch einen dezentral erarbeiteten Konsens mithilfe des Smart Contracts getroffen. Smart Contract Code: <https://etherscan.io/address/0xbb9bc244d798123fde783fcc1c72d3bb8c189413#code>

Bezüglich der Sicherheit des Contracts Codes aus Abschnitt 3.3.2 gibt es 2 Angriffsfaktoren, die berücksichtigt werden müssen. Der Code der payout Funktion befindet sich zur Wiederholung am Ende der Seite.

Wahl der Zufallsquelle

Als Zufallsquelle dürfen keine durch Miner oder andere Teilnehmer manipulierbare Werte genommen werden. Beispielsweise würde die Verwendung des Statements `block.timestamp(payoutBlockNumber)` in Zeile 4 der payout Funktion zu einer wesentlich größeren Angriffsfläche führen. Miner können in einem gewissen Rahmen entscheiden welchen Wert sie in das Timestamp-Feld des Blockheaders schreiben und somit die Ausführung des Codes manipulieren.

Rekursive Auszahlung

Verwendet man das Statement `winnerAddress.call.value(amount)` statt `winnerAddress.transfer(amount)` in Zeile 12 zur Auszahlung, kann ein Angreifer eine mehrfache Auszahlung veranlassen indem er die Adresse eines vorher präparierten Smart Contracts als Auszahlungsadresse angibt. Sobald ein Smart Contract Ether empfängt, ohne dass dabei explizit eine Funktion aufgerufen wird (beispielsweise durch `address.call.value(amount)`), wird die default Funktion des Contracts aufgerufen. Präpariert der Angreifer die default Funktion nun so, dass diese die payout Funktion des Glücksspiel Contracts aufruft, kann er dadurch eine erneute Abarbeitung der payout Funktion bewirken. Da die payout Funktion keinen Schutz gegen eine erneute Abarbeitung besitzt, wird der Topfbetrag ein weiteres Mal auf die Adresse des Gewinners überwiesen.¹⁶ Als Schutzmaßnahme ist es ratsam das Statement `potClosed = false` aus Zeile 15 an den Anfang der Funktion (beispielsweise in Zeile 4) zu verschieben, um einen rekursiven Aufruf zu verhindern.

Die Solidity Dokumentation enthält eine Reihe von Beispielen [19], die die Sicherheit von Smart Contracts betreffen. Entwickler sollten sich dieser bewusst sein, bevor sie einen Smart Contract veröffentlichen der Geld verwaltet.

```

1 function payout() public{
2     assert(potClosed);
3     assert(block.number>payoutBlockNumber);
4     payoutBlockHash = block.blockhash(payoutBlockNumber);
5     if(payoutBlockHash == 0){
6         nbrOfMissedPayouts++;
7     } else {
8         winner = uint256(payoutBlockHash) % NBR_OF_SLOTS;
9         address winnerAddress = payoutAddresses[winner];
10        uint amount= EXPECTED_POT_AMOUNT*NBR_OF_SLOTS;
11        amount +=
            EXPECTED_POT_AMOUNT*NBR_OF_SLOTS*nbrOfMissedPayouts;
12        winnerAddress.transfer(amount); // send pot amount to
            winner
13        nbrOfMissedPayouts = 0;
14    }
15    potClosed = false;
16    nbrOfParticipants=0;
17 }
```

¹⁶Dies funktioniert nur, falls der Glücksspiel Contract genügend Ether besitzt. Außerdem muss der Angreifer einen Abbruchmechanismus in seiner default Funktion vorsehen.

Kapitel 4

Sonstige Distributed-Ledger-Technologie

4.1 Directed Acyclic Graph

Bei einem DAG handelt es sich um einen gerichteten azyklischen Graphen, der im Bereich der Distributed-Ledger-Technologie dazu eingesetzt wird Transaktionen zu speichern. Die Kryptowährung IOTA¹ setzt solch eine Datenstruktur ein. Der Konsens entsteht nicht durch eine Blockchain auf die sich alle Teilnehmer mithilfe des Konsensalgorithmus einigen. Der Konsens entsteht dadurch, dass Teilnehmer neue Transaktionen nur auf Transaktionen aufbauen, die sie für gültig halten. Um eine Transaktion in den Graphen zu schreiben, muss der Absender Proof-of-Work erledigen.

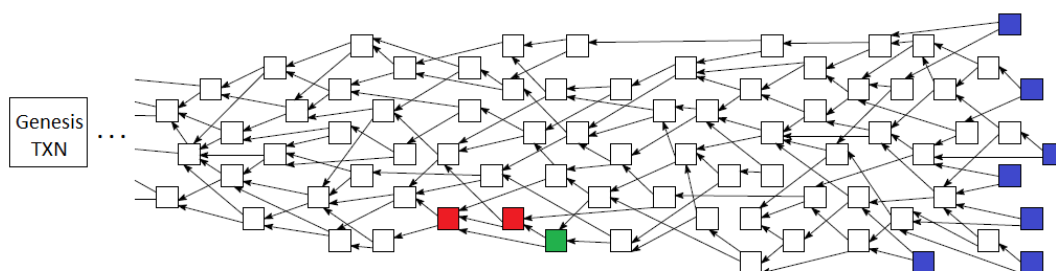


ABBILDUNG 4.1: Directed Acyclic Graph [33]

Die grüne Transaktion aus Abbildung 4.1 referenziert die beiden roten Transaktionen und bestätigt sie auf diese Art und Weise. Je tiefer eine Transaktion im Graphen steckt, desto mehr Proof-of-Work baut auf ihr auf. Am rechten Rand befinden sich neue, noch unbestätigte Transaktionen.

Kryptowährungen auf Basis solch einer Datenstruktur sind nicht für die Gewinnerauswahl einer Glücksspielanwendung nutzbar, da man sich nicht bereits im Vorfeld auf ein eindeutiges, in der Zukunft liegendes, aber dennoch zufälliges Ereignis einigen kann. Weiterführende Informationen zu IOTA und eine genaue Beschreibung der DAG Datenstruktur findet man unter [33].

¹<https://www.iota.org/>

4.2 Proof of Stake

Proof of Stake bezeichnet einen Konsensalgorithmus mit dem ein Blockchain-Netzwerk sich durch einen gewichteten Zufallswert darauf einigt, welcher Teilnehmer den nächsten Block erzeugen darf. Je höher die Anzahl Kryptowährungseinheiten (der sogenannte Stake²) eines Teilnehmers, desto höher ist die Wahrscheinlichkeit, dass dieser den nächsten Block bestimmen darf. Solch ein Abstimmungsprozess ist deutlich ressourcenschonender, da die beim Mining(Proof-of-Work) anfallenden Strom- und Hardwarekosten wegfallen. Proof of Stake hat jedoch auch einige Nachteile und ist deutlich komplexer zu implementieren. Beispielsweise ist die zufällige Auswahl des Teilnehmers, der den nächsten Block erzeugen darf, kein triviales Problem. Es gibt eine Reihe an unterschiedlichen Implementierungen des Proof of Stake Konsensalgorithmus. Die Kryptowährungen Cardano [15], NXT [21] und Peercoin [22] verwenden Formen von Proof of Stake. Eine Reihe von Vor- und Nachteilen von Proof of Stake sowie eine Beschreibung des Nothing-at-Stake Angriffs³ findet sich in [8].

Ob der Zufallswert, der bei Proof of Stake den nächsten Blockerzeuger auswählt, auch für die zufällige Gewinnerauswahl einer Glücksspielanwendung geeignet ist, hängt von der jeweiligen Implementierung ab und muss daher gesondert analysiert werden.

4.3 Second Layer Solutions

Eine Möglichkeit Transaktionsgebühren zu sparen bieten sogenannte Second Layer Solutions. Dies sind Protokolle und Plattformen, die es ermöglichen den Base Layer (die Blockchain) möglichst wenig zu belasten indem Transaktionen nur in die Blockchain geschrieben werden, wenn es nicht anderes möglich ist. Ein Handelsplatz auf dem man Bitcoins an- und verkaufen kann, ist ein Beispiel für eine Plattform als Second Layer Solution. Alle Transaktionen zwischen den Kunden, innerhalb des Handelsplatzes, berühren nicht die Blockchain. Es werden ausschließlich Transaktionen in die Blockchain geschrieben, wenn Kunden in den Handelsplatz ein- und auszahlen. In dieser Art der Second Layer Solution nimmt der Handelsplatz die Rolle einer Trusted Third Party ein. Das Lightning Netzwerk Protokoll ist hingegen eine Second Layer Solution, die sogenannte Payment Channels verwendet, um vollständig ohne Trusted Third Party auszukommen.

4.3.1 Payment Channel

Ein Payment Channel ist eine Technik, die es erlaubt eine Reihe von Zahlungen zwischen zwei Parteien auszuführen, ohne dass dabei alle Zahlungstransaktionen in die Blockchain geschrieben werden müssen. Lediglich das Öffnen und Schließen eines Payment Channels erfordert Transaktionen, die in die Blockchain geschrieben werden. Alle Transaktionen auf die sich beide Teilnehmer zwischen der öffnenden und der schließenden Transaktion einigen, berühren nicht die Blockchain und werden lediglich Off-Chain ausgetauscht. Eine genaue Beschreibung, wie Payment Channels

²Der Stake ist somit der Anteilsnachweis, der dem Teilnehmer einen Anreiz verleihen soll, sich der Konsensregeln entsprechend zu verhalten. Je nach Implementierung fließen auch noch andere Werte wie beispielsweise die Teilnahmedauer (Coin-Age) in die Berechnung ein.

³Dieser Angriff beschreibt das Problem, dass Teilnehmer an mehreren Forks der Blockchain gleichzeitig arbeiten können, ohne dass dadurch für sie Kosten wie beim Proof-of-Work Mining anfallen.

technisch funktionieren und warum keine Partei die jeweils andere betrügen kann, findet man in [32].

4.3.2 Lightning Netzwerk

Das Lightning Netzwerk umfasst zurzeit circa 2000 Netzwerkknoten, die durch ungefähr 6000 Payment Channel miteinander verbunden sind. Es erlaubt Off-Chain Zahlungen zwischen den Teilnehmern, auch wenn diese nicht direkt über einen Payment Channel miteinander verbunden sind.⁴

Die Bitcoin Glücksspielanwendung aus Kapitel 2 könnte Off-Chain Transaktionen über das Lightning Netzwerk für Ein- und Auszahlungen akzeptieren. Dies würde die Transaktionskosten für die Teilnehmer deutlich reduzieren. Off-Chain Transaktionen sind allerdings nicht für alle Teilnehmer des Spiels nachprüfbar, da sie nicht in die Blockchain geschrieben werden. Die Anforderung der transparenten Ein- und Auszahlungen wäre somit verletzt.

⁴Dies funktioniert unter der Voraussetzung, dass ein Pfad aus Payment Channel existiert, der beide Teilnehmer miteinander verbindet.

Kapitel 5

Fazit

Diese Arbeit hat am Beispiel einer Glücksspielanwendung gezeigt, dass der Einsatz von Blockchain-Technologie es anonymen, sich gegenseitig nicht vertrauenden Parteien ermöglichen kann, direkt miteinander zu interagieren, ohne dass dabei eine Trusted Third Party als Mittelsmann verwendet werden muss. Das vorher benötigte Vertrauen wurde in ein öffentliches, transparentes System verlagert, dessen inhärente Spieltheorie den Teilnehmern finanzielle Anreize liefert sich korrekt zu verhalten.

Diese Arbeit hat zunächst gezeigt, wie man solche Systeme in eine eigene Anwendung integrieren kann und auf welche Besonderheiten dabei zu achten ist. Außerdem wurde aufgezeigt, dass man Systeme, die auf einem Proof-of-Work Konsensalgorithmus basieren, in einem gewissen Rahmen als eine verlässliche Zufallsquelle nutzen kann. Die im ersten Ansatz entwickelte, auf der Bitcoin Blockchain aufbauende Glücksspielanwendung erlaubt es dem Nutzer die zufällige Gewinnerauswahl nachzuprüfen. Die Anwendung kann den Nutzer in dieser Hinsicht nicht benachteiligen oder betrügen. Lediglich die von der Anwendung vorzunehmende Auszahlungstransaktion an den Gewinner bietet eine gewisse Angriffsfläche. Der Endnutzer muss der Anwendung vertrauen, dass diese die Auszahlung korrekt ausführt. Eine korruptierte, sich falsch verhaltende Anwendung fällt dem Endnutzer allerdings auf. Diese Problematik wurde mithilfe sogenannter Smart Contracts der Ethereum Blockchain gelöst. Diese erlauben es dem Nutzer vollständig auf Vertrauen verzichten zu können, da die Geschäftslogik der Glücksspielanwendung in der Blockchain verankert ist und von allen Teilnehmern des Netzwerks ausgeführt wird. Durch den Einsatz der Ethereum Blockchain wurde das Ziel, vollständig auf eine Trusted Third Party zu verzichten, erreicht.

Das Beispiel der Glücksspielanwendung auf der Ethereum Plattform ist in soweit gelungen, da keine zusätzlichen Daten aus der echten Welt für diesen Anwendungsfall benötigt wurden. Die gesamte Interaktion findet innerhalb des Ethereum Netzwerks statt und wird durch den bewährten Einsatz von Kryptographie abgesichert. Andere Anwendungsfälle für Blockchain-Technologie wie beispielsweise Supply-Chain-Management oder Logistik und Transport sind schwieriger zu realisieren, da man auf Daten aus der echten Welt angewiesen ist. An der Schnittstelle, die diese Daten in die Blockchain schreibt und somit dort verfügbar macht, ist es unmöglich vollständig auf Vertrauen zu verzichten.

Ein weiterer offene Frage ist, ob es Blockchains in Zukunft schaffen werden, ausreichend zu skalieren. Sogenannte Second Layer Solutions bieten einen Ansatz valide Transaktionen außerhalb der Blockchain zu akzeptieren und nur den resultierenden Zustand in die Blockchain zu schreiben. Ob dies ausreicht, die eigentliche Blockchain genügend zu entlasten, damit On-Chain Transaktionen langfristig bezahlbar bleiben, kann nur die Zeit zeigen.

Anhang A

Vorhandene Glücksspielseiten

A.1 Bitcoin

Die Internetseite Crypto Games [10] bietet ein Würfelspiel an, bei dem der Nutzer mit Kryptowährungen bezahlen kann. Der Spieler wettet darauf, dass der Wert einer "zufällig" generierten Zahl (zwischen 0 und 100) über einem bestimmten Zielwert liegt. Der Spieler kann nachdem die Wette platziert ist eigenständig prüfen, ob er gewonnen hat.

ABBILDUNG A.1: MANUAL BET

Das Formular aus Abbildung A.1 lässt den Spieler die Höhe des Einsatzes und den Multiplikator anpassen. Je höher der Multiplikator, desto höher passt die Glücksspielseite den Zielwert an.

Bereits bevor der Spieler die Wette abschließt, teilt ihm die Seite den SHA256 Hash des sogenannten "Server seed" mit. Der Server Seed ist zu diesem Zeitpunkt nur dem Service bekannt und wird erst nach dem Abschluss der Wette veröffentlicht. Außerdem ermöglicht die Seite es dem Spieler den sogenannten "Next client seed" frei zu wählen. Dieser geht zusammen mit dem Server Seed in die Berechnung der Gewinnerauswahl ein. Sobald der Spieler die Wette abschließt, veröffentlicht der Service den "Server seed". Der Spieler kann durch die Berechnung des SHA256 Hash nachprüfen, ob es sich wirklich um den echten Wert handelt.

MANUAL BET AUTO BET HOW TO PLAY PROVABLY FAIR

Last server seed SHA256: e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855

Last server seed:

Last client seed:

Next server seed SHA256: 97aef9d49aed332b1893d982072d626714b359977ad8a46624e0168bdc232c

Next client seed: Q4tF5RbBWmkuB6gia2ktw4Gi3IT1uoi30a05V9b **RANDOMIZE**

☐ I understand the risks, I want to manually change my client seed

ABBILDUNG A.2: PROVABLY FAIR

Nun erfolgt die Gewinnerauswahl. Zunächst wird der SHA512 Hash des konkatinierten Server und Client Seeds berechnet. Dieser Hash Wert liefert die Zufallsquelle für die Gewinnerauswahl. Die ersten 5 Stellen des Hashs in Hexadezimaldarstellung werden in eine Dezimalzahl konvertiert. Anschließend werden die letzten 5 Ziffern als Zahl zwischen 0 und 100 mit 3 Nachkommastellen betrachtet. Die nachfolgende Abbildung veranschaulicht diesen Prozess.

Dice bet 2,234,848,459

Player: Birch523

Bet ID: 2,234,848,459 Date and time: 19/05/2018 06:58:07

Bet amount: 0.00000640 BTC

Target: > 50.399 Multiplier: 2.00000x

Result: 72.846

Profit: 0.00000640 BTC

Server seed: 6loJ990f76iF9sECWiiwIBxrB54iPI8wYTwyL5g

Client seed: r8Z4Ve6aPxI0FNQu9yWTB9TXMJ1qdfQZm9v0eUan

SHA512 hash: 8bdaec0c4c4b9bfc608590c7007a43f6ba7f378b6c0488af98df36521ea6ef10c6b45bcbf469446261d3e809aad788df5767b88a

Step:	Hex:	Decimal:	Last digits:	Result:
1	8bdae	572846	72.846	72.846

ABBILDUNG A.3: BET RESULT

Im Beispiel aus Abbildung A.3 liegt die resultierende Zahl (72,846) über dem Zielwert 50,399. Dies führt dazu, dass der Spieler gewinnt und seinen Einsatz ausgezahlt bekommt.

A.2 Ethereum

Die Internetseite [39] bietet ein Würfelspiel an, das durch einen Smart Contract auf der Ethereum Plattform umgesetzt ist. Die Internetseite übernimmt dabei lediglich die Visualisierung der platzierten Wetten. Die Teilnahme benötigt keinen Account, da man ausschließlich mit dem Smart Contract¹ interagiert. Statt des Blockhashs wird ein sogenannter Oracle Service [26] zur Gewinnerauswahl verwendet. Dieser liefert Zufallszahlen des Services [24] zusammen mit einer Echtheitsgarantie (Signatur) innerhalb von Transaktionen an den Smart Contract. Der Smart Contract prüft die Echtheit der Daten mittels der Signatur und führt anschließend die Gewinnerauswahl und Auszahlung durch. Detaillierte Informationen über die Funktionsweise eines Oracles und dessen Einsatzmöglichkeiten findet man unter [7].

¹<https://etherscan.io/address/0xdd98b423dc61a756e1070de151b1485425505954#code>

Quellenverzeichnis

- [1] Andreas M. Antonopoulos. *Mastering Bitcoin*. O'Reilly, 2015. URL: <https://github.com/bitcoinbook/bitcoinbook> (besucht am 03.03.2018).
- [2] *Bitcoin Full Node API*. 2015. URL: <http://chainquery.com/bitcoin-api> (besucht am 09.02.2018).
- [3] *Bitcoin Improvement Proposal 21*. 2013. URL: <https://github.com/bitcoin/bips/blob/master/bip-0021.mediawiki> (besucht am 03.03.2018).
- [4] *bitcoinj*. 2011. URL: <https://bitcoinj.github.io/> (besucht am 09.02.2018).
- [5] *Blockchain Info Bitcoin Explorer*. 2011. URL: <https://blockchain.info/> (besucht am 09.02.2018).
- [6] Vitalik Buterin. *A Next-Generation Smart Contract and Decentralized Application Platform*. Nov. 2013. URL: <https://github.com/ethereum/wiki/wiki/White-Paper> (besucht am 03.03.2018).
- [7] Vitalik Buterin. *Ethereum and Oracles*. 2014. URL: <https://blog.ethereum.org/2014/07/22/ethereum-and-oracles/> (besucht am 19.05.2018).
- [8] Vitalik Buterin. *Proof of Stake FAQ*. 2016. URL: <https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ> (besucht am 22.05.2018).
- [9] *Clique PoA protocol and Rinkeby PoA testnet*. 2017. URL: <https://github.com/ethereum/EIPs/issues/225> (besucht am 16.03.2018).
- [10] *Crypto Games*. 2014. URL: <https://www.crypto-games.net/> (besucht am 09.02.2018).
- [11] *Deconstructing the DAO Attack*. 2016. URL: <http://vessenes.com/deconstructing-the-dao-attack-a-brief-code-tour/> (besucht am 31.03.2018).
- [12] Igor Drobiazko. *Tapestry 5: Die Entwicklung von Webanwendungen mit Leichtigkeit*. Pearson Deutschland, 2010.
- [13] *Ethereum Improvement Proposal 681*. 2017. URL: <https://github.com/ethereum/EIPs/blob/master/EIPS/eip-681.md> (besucht am 10.03.2018).
- [14] *ethereumj*. 2016. URL: <https://github.com/ethereum/ethereumj> (besucht am 15.03.2018).
- [15] Cardano Foundation. *Cardano*. 2018. URL: <https://www.cardano.org/en/home/> (besucht am 22.05.2018).
- [16] Ethereum Foundation. *Ethereum Official Website*. 2018. URL: <https://www.ethereum.org/> (besucht am 09.05.2018).
- [17] Ethereum Foundation. *Solidity Dokumentation*. 2018. URL: <https://solidity.readthedocs.io/en/v0.4.0/> (besucht am 09.05.2018).
- [18] Ethereum Foundation. *Solidity Global Variables*. 2018. URL: <http://solidity.readthedocs.io/en/develop/units-and-global-variables.html> (besucht am 09.05.2018).

- [19] Ethereum Foundation. *Solidity Security Considerations*. 2018. URL: <https://solidity.readthedocs.io/en/develop/security-considerations.html> (besucht am 09.05.2018).
- [20] Litecoin Foundation. *Litecoin Official Website*. 2018. URL: <https://litecoin.com/> (besucht am 09.05.2018).
- [21] Nxt Foundation. *Nxt*. 2013. URL: <https://nxtplatform.org/> (besucht am 22.05.2018).
- [22] Peer Coin Foundation. *Cardano*. 2012. URL: <https://peercoin.net/> (besucht am 22.05.2018).
- [23] Brian Neil Levine George Bissias. *Bobtail: An adjustment to proof of work that reduces variance and improves security*. 2017. URL: <https://scalingbitcoin.org/stanford2017/Day1/bobtail.Bitcoin-scaling-2017.key.pdf> (besucht am 21.04.2018).
- [24] Dr Mads Haahr. *Random number generater API*. 1998. URL: <https://www.random.org/> (besucht am 19.05.2018).
- [25] Georg Hoever. *Kryptologie*. 2016. URL: www.hoever-downloads.fh-aachen.de/krypto/KryptoSkript.pdf (besucht am 04.06.2018).
- [26] Oraclize Limited. *Ethereum Oracle Service*. 2015. URL: <http://www.oraclize.it/> (besucht am 19.05.2018).
- [27] Piper Merriam. *Ethereum Alarm Clock Contract*. 2018. URL: www.ethereum-alarm-clock.com/ (besucht am 09.05.2018).
- [28] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Nov. 2008. URL: <https://bitcoin.org/en/bitcoin-paper> (besucht am 03.03.2018).
- [29] Jonathan Otto. *Bitcoin Orphaned Blocks*. 2018. URL: <https://blockchain.info/orphaned-blocks> (besucht am 05.05.2018).
- [30] Jonathan Otto. *Understanding Bitcoin Transactions*. 2018. URL: <https://www.jonathanotto.com/bitcoin-transactions> (besucht am 05.05.2018).
- [31] *Peer-to-Peer*. 2018. URL: <https://de.wikipedia.org/wiki/Peer-to-Peer> (besucht am 01.04.2018).
- [32] Joseph Poon und Thaddeus Dryja. *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. Jan. 2016. URL: <https://lightning.network/lightning-network-paper.pdf> (besucht am 20.05.2018).
- [33] Serguei Popov. *The tangle*. 2016. URL: <https://www.iota.org/research/academic-papers> (besucht am 21.04.2018).
- [34] Monero Project. *Monero Official Website*. 2018. URL: <https://getmonero.org/> (besucht am 09.05.2018).
- [35] Frank Stajano und Richard Clayton. „Cyberdice: Peer-to-Peer Gambling in the Presence of Cheaters“. In: *Lecture Notes in Computer Science* 62.1 (Jan. 2011), S. 1–20. URL: https://link.springer.com/chapter/10.1007/978-3-642-22137-8_9.
- [36] Blockchain Luxembourg S.A. *Blockchain Info Bitcoin Hashrate*. 2018. URL: <https://blockchain.info/de/charts/hash-rate> (besucht am 17.05.2018).
- [37] Blockchain Luxembourg S.A. *Blockchain Info Bitcoin Mining Pools*. 2018. URL: <https://blockchain.info/de/pools> (besucht am 10.05.2018).

- [38] Conor Svensson. *Web3j Commandline Tools Dokumentation*. 2018. URL: https://docs.web3j.io/command_line.html (besucht am 09.05.2018).
- [39] *vDice Ether Games*. 2016. URL: <https://www.vdice.io/> (besucht am 09.02.2018).
- [40] *web3j*. 2016. URL: <https://github.com/web3j/web3j> (besucht am 15.03.2018).
- [41] Sam Wouters. *Why Schnorr signatures will help solve 2 of Bitcoin's biggest problems today*. 2017. URL: <https://medium.com/@SDWouters/why-schnorr-signatures-will-help-solve-2-of-bitcoins-biggest-problems-today-9b7718e7861c> (besucht am 10.03.2018).

Abbildungsverzeichnis

2.1	Client-Server Peer-to-Peer [31]	5
2.2	Bitcoin Zustandsveränderung durch Transaktion [6]	6
2.3	Kette von Blöcken	6
2.4	Schritt 1	10
2.5	Schritt 2	11
2.6	Schritt 3	11
2.7	Schritt 4	11
2.8	Schritt 5	12
2.9	Schritt 6	12
2.10	Schritt 7	12
2.11	Schritt 8	13
2.12	Schritt 9	13
2.13	Schritt 10	13
2.14	Schritt 11	14
2.15	Schritt 12	14
2.16	Bitcoin Core: Full-Node Aufbau [1]	15
2.17	Blockheader Kette [6]	16
2.18	Glücksspielanwendung Aufbau und Interaktion	17
2.19	Datenmodel Klassendiagramm	18
2.20	Geschäftslogik Klassendiagramm	19
2.21	Leerer Topf	26
2.22	Smartphone Überweisungsformular	27
2.23	Zahlungsbestätigung	27
2.24	Transaktion empfangen	28
2.25	Spieler zu Topf hinzugefügt.	28
2.26	Topf geschlossen	29
2.27	Gewinner ermittelt	29
2.28	Auszahlung beendet	30
2.29	Verteilung der SHA256 Hashfunktion	34
2.30	Bitcoin Mining Pool Hash Rate [37]	36
2.31	Gleichung: $E = 100000 * (1 - h)$	37
2.32	Auszahlungstransaktion Details	38
2.33	Auszahlungstransaktion Inputs und Outputs	38
2.34	Auszahlungstransaktion Skripts	39
2.35	Bitcoin Fork	40
3.1	Sequenzielle Veränderung des Systemzustands [6]	43
3.2	Veränderung des Systemzustands Beispiel [6]	44
3.3	Smart Contract Automat	47
3.4	Ethereum: Netzwerk Integration	48
3.5	Klassendiagramm Web3J	52
3.6	Klassendiagramm Ethereum	54

3.7	Leerer Topf	59
3.8	My Ether Wallet	60
3.9	Liste aller Smart Contract Funktionen	60
3.10	Aufruf der EXPECTED_POT_AMOUNT Funktion	61
3.11	Aufruf der deposit Funktion	61
3.12	Eingang der ersten Zahlung	62
3.13	Topf geschlossen	62
3.14	Aufruf der payout Funktion	63
3.15	Gewinner ausgewählt	63
3.16	Block Explorer: Smart Contract	64
3.17	Block Explorer: Payout Transaktion	64
3.18	Verteilung der Keccak-256 Hashfunktion	67
4.1	Directed Acyclic Graph [33]	69
A.1	MANUAL BET	73
A.2	PROVABLY FAIR	74
A.3	BET RESULT	74