BLOCKCHAIN DECRYPTED

Wettbewerbsarbeit

Georgette Weingärtner, Kantonsschule Baden, G4a

2019

Abstract

Die Begriffe Bitcoin und Blockchain finden sich momentan oft in Schlagzeilen der Zeitungen und sind ein sehr beliebtes Diskussionsthema. Jedoch wissen nicht viele, was tatsächlich hinter diesen Begriffen steckt. Diese Tatsache motivierte uns dazu, die Themen genau zu verstehen und auf eine verständliche und spannende Art darzustellen, sodass sie leicht nachvollzogen werden können. Unsere Maturaarbeit beschreibt den Aufbau und die Funktionsweise der Blockchain. Die Grundlagen der modernen Blockchain Technologie werden im Zuge der Arbeit ausführlich erklärt. Dabei werden auch die mathematischen Aspekte der Blockchain, über welche sonst nur wenig Informationen zu finden sind, ausführlich dargestellt. Schlussendlich werden alle Grundlagen schrittweise zum grossen Bild der Blockchain zusammengesetzt. So kann gut verstanden werden, wie diese überaus spannende Technologie funktioniert. Weiterhin wird gezeigt, dass die Blockchain weitaus mehr spannende und zukunftsrelevante Anwendungen hat als das klassische Beispiel des Bitcoins.

Um die Anwendungen der Blockchain noch tiefgründiger zu verstehen, wurde im Rahmen dieser Maturaarbeit ein eigenes Programm auf einer Blockchain entwickelt. Hierfür wurde in einem ersten Schritt die auf Blockchains verwendete Programmiersprache Solidity erlernt und vertieft. Mit dieser Grundlage wurde dann ein TicTacToe-Spiel auf der Blockchain umgesetzt. In das Spiel sind viele verschiedene Aspekte der Blockchain integriert. Auf der einen Seite läuft das Programm auf einer Blockchain ab. Damit ist es möglich weiter auszuführen, wie auf der Blockchain programmiert werden kann. Auch kann dadurch besser verstanden werden, was die entscheidenden Vorteile vom Programmieren auf Blockchains sind. Zusätzlich wurde mit Hilfe eines JavaScript-Programmes eine Schnittstelle zwischen dem Programm auf der Blockchain und einer Html-Website erstellt. Im Programm ist sogar eine eigene Kryptowährung mit dem Namen „Kanti-Coin“ integriert. Beim Ausführen des Programms, also beim Spielen, erhält der Gewinner vom Verlierer am Ende des Spiels einen Betrag dieser Kryptowährung. Die eigene Kryptowährung zeigt den Aspekt der Transaktion von Währungen auf der Blockchain sehr schön auf.

Zielpublikum, Voraussetzungen und Form

Die Arbeit ist interessant für alle, die sich gerne intensiver mit dem Thema Blockchain auseinandersetzen wollen und daran interessiert sind, welche Themen die Informatik die nächsten Jahre voraussichtlich stark prägen werden. Sie ermöglicht es zu verstehen, wieso die Blockchain so zukunftsrelevant ist und was sie ausser dem Bitcoin zu bieten hat. Für die mathematischen Grundlagen ist ein Basiswissen in der Mathematik nötig, was die mathematischen Abschnitte besonders für gebildete Laien interessant macht. Die verwendeten mathematischen Grundlagen werden jeweils im Vorhinein zusammenfassend beschrieben, sodass die mathematischen Abschnitte zum Thema Blockchain mit Basis-Kenntnissen in der Mathematik ohne Probleme nachvollziehbar sind. Fragen zu jedem Kapitel sollen das Verständnis des jeweiligen Kapitels fördern und bei der Selbstüberprüfung des Gelernten helfen.

Betreuer

Armin P. Barth (Erstbetreuer)

Kurt Doppler (Zweitbetreuer)

Erstellt von März bis November 2018

INHALTSVERZEICHNIS

1. EINLEITUNG

1.1. Einführung

Die Ursprünge des Bankenwesens sind bereits im 2. Jahrhundert vor Christus im mesopotamischen Reich zu finden. Schon damals waren Händler darauf angewiesen, sich gegenseitig Geld zu leihen, um den reibungslosen Ablauf des Handelns sicherzustellen. Es wurden auch schon meist Zinsen auf das geliehene Geld erhoben.

Das Bankenwesen, wie wir es heute kennen, hat seinen Ursprung in Italien. Im 12. Jahrhundert begannen italienische Geschäftsleute Kaufleuten bei Messen Geld auszuleihen. Dieses musste ihnen dann bis zum Messeende zurückbezahlt werden. Im späten Mittelalter entwickelten sich aus diesen Geschäftsleuten dann sesshafte, zertifizierte Bankiers oder Geldwechsler. Sie mussten der Republik schwören nicht zu **stehlen, zu fälschen oder zu betrügen**. Die Echtheit und Qualität der Münzen wurde mit Hilfe von Waagen überprüft, wobei unechte Münzen oder Münzen von schlechter Qualität aus dem Verkehr zu nehmen waren. Wollte jemand Geld überweisen, so notierten die Bankiers in ihren Büchern eine Abbuchung beim Kontostand des Senders und schrieben das Geld dem Konto des Empfängers zu.

Abbildung 1: Banco Del Giro. Dalvenetoalmondo, 10.6.2018

Um 1619 entstand dann in Venedig die erste staatliche Bank, die „Banco del Giro“. [1] [2] [3] Obwohl sich das Bankenwesen mit der Zeit stark verändert hat, haben sich die grundlegenden Aufgaben der Bank von damals bis heute kaum geändert. Sie nehmen die Vermittler- und Überwachungsposition im Geldkreislauf ein.

Die Kunden der Banken legen ihr Geld als Einlagen bei der Bank ab. Für diese Geldeinlagen bezahlt die Bank ihren Kunden Zinsen. Das erhaltene Geld bezahlt die Bank dann wieder an andere Kunden in Form von Krediten aus und erhält darauf wiederum Zinsen. Um einen Gewinn zu erzielen, verlangt die Bank von ihren Kunden mehr Zinsen für Kredite, als sie den Kunden für ihre Einlagen ausbezahlt. Durch diese Differenz, auch Zinsspanne genannt, erzielen Banken ihre Gewinne.

[1] 1500 Entwicklung des Bankenwesens, heruntergeladen am 10.6.2018
[2] Keuper: Bankwesen Oberitalien, heruntergeladen am 10.6.2018
[3] Banken, in: Historisches Lexikon der Schweiz, heruntergeladen am 10.6.2018

Weiterhin sind Banken für das Durchführen und Protokollierungen von Überweisungen von einem Konto auf ein anderes zuständig. Bei den meisten Überweisungen verlangen die Banken wiederum einen kleinen Betrag. Auch für die Führung eines Kontos, also für das zuverlässige Protokollieren der Ein- und Abgänge, ist die Bank verantwortlich und es werden Gebühren verlangt.

Selbstverständlich übernehmen Banken zudem viele weitere Aufgaben, wie zum Beispiel Kunden bezüglich Investitionen und Geldanlagen zu beraten und selbst mit Wertpapieren zu handeln. Darauf wollen wir hier aber nicht näher eingehen.

Die Gebühren auf Kontoführung und Überweisungen und die Zinsspanne verlangen die Banken für die Dienstleistung, da sie als Vermittler im Geldkreislauf zwischen einzelnen Privatpersonen aber auch Unternehmen wirken. Institutionen, welche eine Vermittlerposition zwischen verschiedenen Parteien einnehmen, werden auch als **Mittelsmänner** bezeichnet. [4] [5]

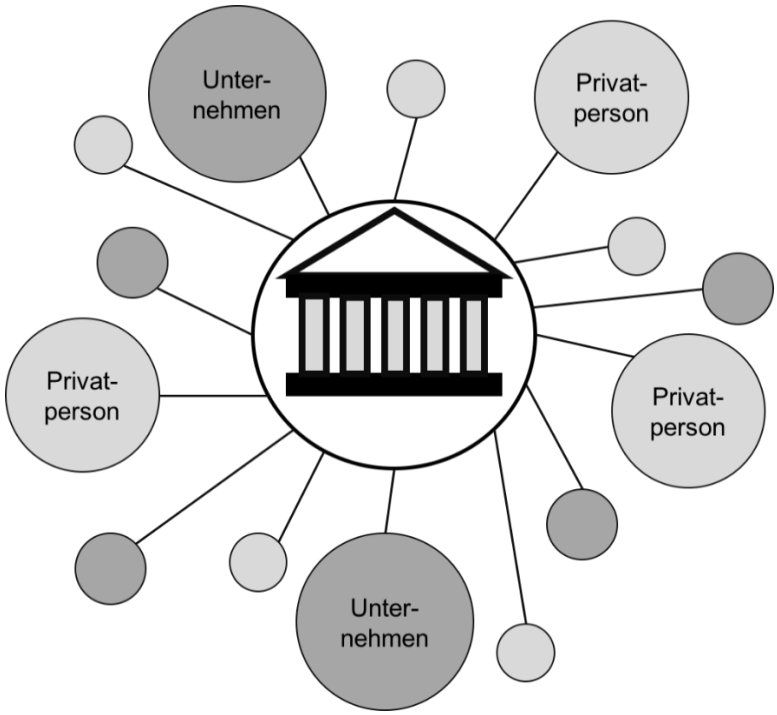

Abbildung 2: Die Bank als Mittelsmann

In der oben abgebildeten Grafik ist sehr schön zu sehen, wie die Bank als Mittelsmann zwischen vielen Privatpersonen aber auch Unternehmen dient. Die Verbindungslinien zeigen hierbei Abhängigkeiten in Form von Geldflüssen. Es existiert ein Netzwerk zwischen den Parteien und der Bank als zentrales Element.

Solche Mittelsmänner existieren nicht nur im Bankenwesen. Ein weiteres Beispiel hierzu ist die Stromversorgung. Der in den Kraftwerken produzierte Strom wird über Elektrizitätswerke an die einzelnen Haushalte verteilt. Immer mehr Haushalte produzieren selber Strom. Teilweise ist hierbei die Stromproduktion höher als der Strombedarf. Der überschüssige Strom dieser Haushalte wird dann weiterhin vom Elektrizitätswerk an Haushalte mit höherem Strombedarf als Stromproduktion verteilt. Der Strompreis, welcher von den Haushalten bezahlt werden muss, ist hierbei viel höher als derjenige, mit dem der überschüssige Strom von Haushalten abgekauft wird. Mit dieser Differenz wird der Mittelsmann bezahlt.[6]

[4] Aufgaben der Bank, in: Jugend und Bildung, heruntergeladen am 10.6.2018
[5] Gebührentarif, in: Alternative Bank Schweiz, heruntergeladen am 10.6.2018
[6] Build Decentralized Energy Communities Secured by the Blockchain, in: HivePower, heruntergeladen am 10.6.2018

Die Bankiers aus dem 12. Jahrhundert mussten einen Eid ablegen kein Geld zu veruntreuen, zu fälschen oder zu betrügen. Auch heutzutage vertrauen wir darauf, dass die Bank eine ehrliche Institution ist, uns nicht betrügt und unsere Transaktionen ordnungsgemäss durchführt und dokumentiert. In der Schweiz wird dies durch die FINMA (Finanzmarktaufsicht) kontrolliert.

Das Vertrauen in die Mittelsmänner ist eine Voraussetzung dafür, dass das Netzwerk der involvierten Parteien funktioniert. Der Mittelsmann ist das zentrale Element, das garantiert, dass die Transaktionen aller Parteien korrekt ablaufen.

Ein Nachteil dieses zentralen Elements ist die Macht, welche es hat. Es muss dringend vertrauenswürdig sein. Anderenfalls könnte es alle, die ihm vertrauen, ausnützen und hintergehen. Es kann ein kompletter Ausfall resultieren oder grosse individuelle Verluste können entstehen.

Weiterhin ist Sicherheit ein grosses Problem. Wird das zentrale Element aus einem Grund von einer fremden, betrügerischen Partei eingenommen, so hat diese sofort Kontrolle über das gesamte Netzwerk. Eine Möglichkeit hierfür ist zum Beispiel ein Hacker, welcher in ein Bankensystem eindringt. Nicht zu vernachlässigen ist auch, dass Mittelsmänner für all ihre Leistungen Geld verlangen.

All diese Punkte zeigen auf, dass ein Netzwerk mit einem zentralen Element beziehungsweise Mittelsmann viele Nachteile hat.[7]

Eine mögliche Lösung der Realisierung von Netzwerken ohne Mittelsmänner bietet die Blockchain. Mit Hilfe der Blockchain kann ein System aus fremden Parteien, welches nicht auf dem Vertrauen der Parteien zueinander basiert, kreiert werden. In den nächsten Kapiteln werden wir verstehen, was die Blockchain ist und wie sie diese Aufgabe meistern kann. Hierbei werden wir auf viele Herausforderungen stossen und zeigen, wie diese durch die Blockchain Technologie gemeistert werden können. Immer wieder werden wir mathematisch interessante Aspekte diskutieren und verstehen.

Die berühmteste Implementierung der Blockchain ist der Bitcoin. Wir werden aber sehen, dass die Blockchain zahlreiche weitere interessante und zukunftsrelevante Anwendungen hat.

Weiterhin wurde im Rahmen der Maturaarbeit ein umfangreiches Programm entwickelt, in welchem Blockchain Technologie eingesetzt wurde. Mit Hilfe dieses Programms werden die theoretisch erklärten Aspekte verdeutlicht und auch die konkrete Umsetzung veranschaulicht.

1.2. Die Anfänge der Blockchain

Im vorherigen Abschnitt haben wir über das Bankenwesen diskutiert und dargestellt, dass Banken als Mittelsmänner bei Geldflüssen („Transaktionen“) dienen. Wir haben dies verallgemeinert zu Netzwerken mit Mittelsmännern und aufgezeigt, dass das Vorhandensein von Mittelsmännern den involvierten Parteien zwar den Vorteil der Sicherheit gibt, andererseits aber viele Nachteile hat. Hier setzt die Blockchain an: Die Motivation der Blockchain ist die Abwicklung von Transaktionen ohne Mittelsmänner.

Der Bitcoin ist die erste und berühmteste Anwendung der Blockchain. Er ist auf der ersten funktionstüchtigen, physisch implementierten Blockchain realisiert. Durch den Bitcoin kann man konkret auf einen Mittelsmann bei der Durchführung von Geldtransaktionen verzichten.

Der Bitcoin ist eine sogenannte **kryptographische Währung** oder kurz **Kryptowährung**. Er ist ein digitales Zahlungsmittel, welches mit Hilfe der in der Blockchain realisierten krypto-

[7] McGew: Disadvantages of a Centralized Network, heruntergeladen am 22.8.2018

grafischer Technologien, wie beispielsweise Digitale Signaturen, abgesichert wird. Er ist eine rein virtuelle Währung, es gibt also keine materiellen Bitcoin-Münzen.[8]

2008 wurde im Internet ein Artikel mit Titel: „Bitcoin: A Peer-to-Peer Electronic Cash System" veröffentlicht. Der Autor des Skripts nennt sich „Satoshi Nakamoto", hierbei handelt es sich um ein Pseudonym. Der Artikel beschreibt, was der Bitcoin ist, wie er funktioniert und wieso die Technologie sicher ist.[9] Einige Zeit später, am 12. Januar 2009, wurde dann (virtuell) der erste Bitcoin getauscht. Zum ersten Mal wurde mit dem Bitcoin im Februar 2010 eine reale Zahlung durchgeführt. Ein Amerikaner erwarb zwei Pizzen für genau 10'000 Bitcoins. In der Spitzenzeit des Bitcoins hatten 10'000 Bitcoins einen Wert von über 180 Millionen Franken – eine ziemlich teure Pizza also.[10] [11]

Der Begriff Bitcoin ist aktuell in aller Munde. Er füllt Schlagzeilen von Zeitungen und Nachrichtensendungen. Sogar der Rapper Eminem hat den Begriff „Bitcoin" in seinem neuen Song „Not Alike" eingebaut: „Now everybody doin' Bitcoin". Besonders beeindruckend ist für viele hierbei natürlich der rasante Anstieg des Wertes von Bitcoin. Es ist jedoch nicht leicht zu verstehen, wie eine Kryptowährung wie der Bitcoin überhaupt funktioniert. Umso grösser ist die Motivation zu verstehen, wieso diese Währung, in welche bereits Milliarden investiert wurden, sicher ist.

Wie bereits erwähnt ist die Blockchain die Technologie, die ermöglicht, dass eine Kryptowährung wie der Bitcoin funktioniert. Bitcoin ist hierbei die Anwendung der Blockchain und darf nicht mit dem Begriff Blockchain gleichgesetzt werden.

Wie die Blockchain das Ersetzen von Mittelsmännern möglich macht, werden wir in dieser Arbeit genau erläutern. Dabei stützen wir uns auf das klassische Anwendungsbeispiel der Blockchain: Transaktionen von einer Person zu einer anderen, welche nicht über einen Mittelsmann ablaufen.

Um die Aufgaben der Blockchain beim Ersetzen des Mittelsmanns zu verstehen beleuchten wir nun systematisch, was eine Bank sicherstellen muss, wenn eine Transaktion von einer Person zu einer anderen durchgeführt wird.

Betrachten wir den Fall, dass ein Kontoinhaber von seinem Konto aus Geld auf ein weiteres Konto überweist.

1. Zuerst muss die Bank sicherstellen, dass der Zahlungsauftrag sicher vom Kontoinhaber und keinem fremden Betrüger in Auftrag gegeben wurde. Dafür muss beispielsweise ein Pin eingegeben werden oder ein amtlicher Ausweis vorgewiesen werden. Dies zeigt auf, ob es sich beim Auftraggeber tatsächlich um den Kontoinhaber handelt, der als einziger berechtigt ist, Geld von seinem Konto aus zu überweisen.
2. Ist dieser Schritt vollendet, führt die Bank die Transaktion durch. Dies bedeutet, dass die Bank von einem Konto einen Geldbetrag abzieht und dann dem Konto des Empfängers gutschreibt. Dabei muss es sich natürlich genau um den Betrag handeln, der vom Kontoinhaber vorher festgelegt wurde.
3. Es darf nicht möglich sein, dass der Auftraggeber im Nachhinein behauptet, er hätte die Transaktion nicht initiiert. Dies wird beispielsweise durch eine Unterschrift oder eine Pineingabe verhindert.

[8] Prof. Dr. Bendel: Kryptowährungen, heruntergeladen am 17.7.18
[9] Nakamoto: Bitcoin, heruntergeladen am 30.5.2018
[10] die Geschichte des Bitcoins, in: moneymuseum, heruntergeladen am 30.5.2018
[11] Bitcoin – Schweizer Franken, in: finanzen.ch, heruntergeladen am 19.9.2018

4. Auch darf niemand später die Transaktion verändern, also weder den überwiesenen Betrag noch das Datum modifizieren. Die Bank protokolliert deswegen die Transaktionen und verändert diese als vertrauenswürdige Partei nicht.
5. Ist ein Betrag einmal von einem Konto auf ein anderes gutgeschrieben worden, darf dieser kein zweites Mal mehr ausgegeben werden. Er darf also nicht zu einem anderen Zweck nochmals verwendet werden. Die Bank setzt dies über den Kontosaldo um, der nach der Transaktion angepasst wird.

Um die Bank als Mittelsmann zu ersetzen, muss die Blockchain all diese Aufgaben sicherstellen. Wie ein solches System ohne Mittelsmann aufgebaut ist, werden wir im folgende genauer betrachten.

1.3. Dezentrales Netzwerk

1.3.1. Wie sieht ein dezentrales Netzwerk aus?

Mit Hilfe des Beispiels der Bank konnten wir verstehen, wie ein Netzwerk mit einem **zentralen Element** aufgebaut ist. Doch wie sieht ein Netzwerk ohne zentrales Element aus?

Um dies zu verstehen, schauen wir uns ein Konzept aus, welches schon lang vor der Blockchain entwickelt worden war:

Im Jahre 500 vor Christus gab es auf der Insel Yap (heute Mikronesien) die Methode, grosse Steinblöcke als Bezahlungsmittel zu benutzen. Ein Block wog hierbei 200 kg und hatte eine Form ähnlich der eines Donuts. Sie wurden „Rai Stones“ genannt. Die Blöcke waren jeweils Eigentum eines Bewohners der Insel, eines Yapis. Wären die Blöcke jeweils im Garten oder nahe dem Haus des Besitzers gestanden, hätte sich ein riesiger Aufwand entwickelt. Jedes Bezahlen mit einem Block wäre dann nämlich damit verbunden gewesen, einen 200 kg Steinblock über die ganze Insel zu transportieren. Deswegen hatten die Yapis ein sehr geschicktes Konzept entwickelt:

Jeder Yapi wusste von allen Rai Stones, wer deren Eigentümer war. Wollte nun der Yapi Bob seinen Rai Stone an die Yapi Alice weitergeben, so teilten sie dies allen anderen Yapis auf der Insel mit. Dies war kein grosser Aufwand, da alle Yapis jeweils miteinander kommunizierten und sich Informationen so schnell verbreiteten.

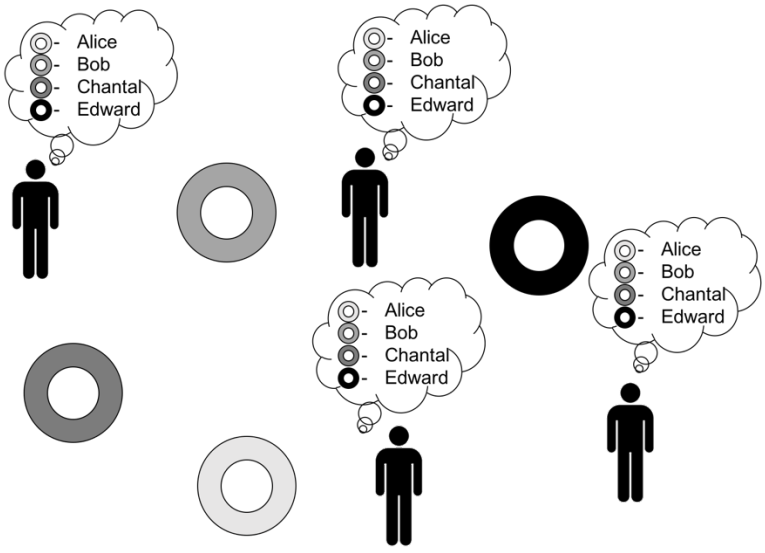

Abbildung 3: Dezentrales Netzwerk unter den Yapis

Wäre ein Yapi unehrlich gewesen und hätte behauptet, dass Alices Rai Stone ihm gehöre, so hätten alle anderen Yapis oder zumindest die Mehrheit der Yapis gewusst, dass er Unrecht hat. Sie konnten so einen Streitfall einfach klären. Hätte ein Yapi von einem Rai Stone vergessen, wem er gehörte, so hätte er ganz einfach die anderen Yapis auf der Insel gefragt. Sie hätten ihm eine Antwort geben können. Wäre ein Stein verloren gegangen oder zerstört worden, hätte sich ebenfalls kein Problem ergeben. Alle Yapis wussten nämlich, wem dieser Stein gehört hatte und konnten ihn einfach in ihrer Vorstellung weiterexistieren lassen.

Der Vorteil dieses Konzeptes war, dass die Besitzverhältnisse nicht von einem einzigen Yapi abhängig waren, sondern von allen Yapis gemeinsam geregelt wurden. Deshalb mussten Änderungen der Besitzverhältnisse – die Transaktionen – nicht bezahlt werden, es fielen also keine Transaktionsgebühren an. Ausserdem gab es keinen Yapi, welcher die Macht über die Besitztümer aller anderen Yapis hatte. [12] [13] [14]

Abbildung 4: Rai Stones auf der Insel Yap. Innermann, 26.10.2018

1.3.2. Peer-to-Peer Netzwerk

Das Konzept der Yapis würde man heute als ein **dezentrales Netzwerk** bezeichnen. Genau solch ein dezentrales Netzwerk wird in der Blockchain ebenfalls verwendet. Stellen wir uns die Situation mit den Yapis bildlich vor. Jeder Yapi ist hierbei ein Knotenpunkt. Jeder Knotenpunkt hat gewisse Informationen. In unserem Beispiel wäre dies die Information, welche Rai Stones im Besitz von wem sind. Die einzelnen Knotenpunkte sind miteinander verbunden. Unsere Yapis beispielsweise sind dadurch, dass sie miteinander reden verbunden. Eine Verbindung bedeutet also einen Austausch von Informationen miteinander.

Bei der Blockchain bestehen die Knotenpunkte aus gleich wichtigen und gleichberechtigten Computern. Die einzelnen Computer stellen jeweils einen Teil ihre Ressourcen wie Rechenleistung, Speicherkapazität und Informationen direkt den anderen Computern im Netzwerk zur Verfügung. Als Verbindung zwischen den einzelnen Computern dient das Internet.

So entsteht ein dezentrales Netzwerk, welches genauer **Peer-to-Peer Netzwerk** genannt wird.[15]

[12] Bower: ancient stone money, heruntergeladen am 26.10.18
[13] Keuper: Steingeld Mikronesien, heruntergeladen am 26.10.18
[14] Nenavath Santosh: YAP's, heruntergeladen am 26.10.18
[15] Peer to Peer Network, in: Lisk, heruntergeladen am 22.08.18

Dezentrale Netzwerke haben Nachteile, denen durch die Blockchain-Technologie Rechnung getragen werden müssen. Im vorherigen Beispiel sind wir davon ausgegangen, dass die Yapis, zumindest grösstenteils, ehrliche Personen sind. Es probiert beispielsweise kein Yapi die anderen Yapis zu bestechen. Ausserdem kennen die Yapis sich gegenseitig und können davon ausgehen, dass die anderen Yapis ihnen im Streitfall weiterhelfen. Wir wollen jedoch, dass theoretisch jeder ein Teil unseres dezentralen Netzwerks sein könnte. Hierfür müssen wir davon ausgehen, dass wir unehrliche Personen in unserem Netzwerk haben, welchen wir nicht vertrauen können. Mit diesem Fakt müssen wir umgehen können, ohne ein zentrales Element zu haben, welches unehrliche Personen erkennt und Betrug verunmöglichen.

Weiterhin dürfen keine technischen Fehler die Sicherheit des dezentralen Netzwerks einschränken. Diesen Teil übernimmt die Technologie der Blockchain, die wir nun immer näher kennenlernen werden.[16]

1.4. Fragen

1. Was ist der Unterschied zwischen Blockchain und Bitcoin?
2. Was sind, anhand des Beispiels der Yapis erklärt, die Vor- und Nachteile eines dezentralen Netzwerks?
3. In einem Peer-to-Peer Netzwerk können Berechnungen schneller durchgeführt werden als ein einzelner Computer dieselben Berechnungen durchführen könnte. Wieso ist dies möglich?

[16] Drescher: Blockchain Basics (Jahr), Kapitel 3f

2. GRUNDLAGEN

2.1. Hashing

2.1.1. Funktionsweise von Hashfunktionen

Einzigartige Merkmale des Menschen sind seit jeher wichtig, um Personen eindeutig und schnell identifizieren zu können. Seit man entdeckt hat, dass jeder Mensch einzigartige Fingerabdrücke hat, werden sie verwendet um beispielsweise Verbrecher zu überführen oder den Zugang zu Tresoren zu sichern. In diesem Kapitel werden wir ein Konzept kennenlernen, mit dem man Daten eindeutig und schnell miteinander vergleichen kann, also sozusagen „identifizieren" kann. Dieses bezeichnet man als digitalen Fingerabdruck. Dieser digitale Fingerabdruck wird in der Blockchain eingesetzt, um deren Sicherheit zu garantieren. Aus diesem Grund ist es wertvoll, zuerst eine genaue Vorstellung von diesem digitalen Fingerabdruck zu gewinnen, um dann alle Aspekte der Funktionsweise der Blockchain zu verstehen.

Der digitale Fingerabdruck wird durch einen speziellen Typ von Funktionen umgesetzt, den **Hashfunktionen**. Eine Hashfunktion wird auf Daten beliebiger Länge angewendet und ordnet jeder beliebig langen Zeichenkette einen Ausdruck mit definierter Länge zu. Die Eingabedaten können hierbei ein Wort, eine Zahl oder auch ein längerer Text sein.

Mit Hilfe eines Fingerabdrucks ist es möglich, sehr effizient einen Menschen mit einer Beschreibung (beispielsweise Täter und Beschreibung des Verdächtigen) zu vergleichen, ohne alle Informationen über diesen Menschen kennen zu müssen. Durch den Einsatz von Hashfunktionen kann man Daten effizient miteinander vergleichen. Dazu wendet man die Hashfunktion auf die zu vergleichenden Daten an, berechnet also deren Hashwert und vergleicht die Hashwerte miteinander. Diese sind von begrenzter Länge und sehr viel effizienter zu vergleichen als die Originaldaten.

Es gibt viele verschiedene Hashfunktionen. Wie lange die generierte Zeichenkette ist, hängt alleine von der Hashfunktion ab. Diejenigen Hashfunktionen, welche am meisten in der Blockchain-Technologie verwendet werden, sind die sogenannten SHA-1 und SHA-256 Funktionen. Dabei steht SHA für Secure Hash Algorithm. Sie wurden vom „National Institutes of Standards and Technology" entwickelt und erfüllen gewisse Sicherheitsstandards, wie zum Beispiel die Kollisionsfreiheit, die wir später vorstellen.[17]

Der SHA-1 bildet jede Zeichenkette auf einen feste Länge von 160 Bit ab, der SHA-256 generiert eine Zeichenkette mit der Länge von 256 Bit.

[17] Secure Hash Algorithm (SHA), in: Techopedia, heruntergeladen am 12.10.2018.
Fingerabdruck aus: http://posad.nl/posad-denkt-aan-jouw-privacy/414px-fingerprint_picture-svg/

2.1.2. Eigenschaften der Hashfunktionen

Hashfunktionen haben charakteristische Eigenschaften. Diese Eigenschaften ähneln denen eines Fingerabdrucks stark und lassen sich deswegen gut damit vergleichen. Wir stellen uns eine Funktion vor, die jedem Menschen einen Fingerabdruck zuordnet.

Der Fingerabdruck jedes Menschen ist individuell: Theoretisch wäre es möglich, dass zwei Menschen denselben Fingerabdruck haben. Da es jedoch so viele Möglichkeiten für Fingerabdrücke gibt, ist die Wahrscheinlichkeit, zwei Menschen mit gleichem Fingerabdruck zu finden, praktisch null.

Der zum Fingerabdruck gehörende Mensch ist nicht ableitbar: Es ist nicht möglich aus einem Fingerabdruck abzuleiten, wie der dazugehörende Mensch aussieht. Die Funktion ist also nicht umkehrbar. Um zu entscheiden, welchem Menschen ein bestimmter Fingerabdruck gehört, muss zuerst von jedem betroffenen Menschen ein Fingerabdruck abgenommen werden und dieser dann mit dem gegebenen Fingerabdruck verglichen werden.

Ein Fingerabdruck von einem Menschen bleibt gleich: Wird von einem Menschen mehrmals der Fingerabdruck bestimmt, so wird dieser stets gleich sein und sich nicht jedes Mal verändern.

Zwei ähnliche Menschen haben einen völlig anderen Fingerabdruck: Eine Ähnlichkeit zweier Menschen bedeutet nicht, dass ihre Fingerabdrücke ebenfalls ähnlich sind. Sogar eineiige Zwillinge, die man kaum unterscheiden kann, besitzen unterschiedliche Fingerabdrücke.

Hashfunktionen besitzen genau die hier beschriebenen Eigenschaften:

Kollisionsfreiheit

Hashfunktionen sind praktisch kollisionsfrei. Dies bedeutet, dass zwei verschiedene Eingabewerte fast immer auf unterschiedliche Ausgabewerte abgebildet werden.

Der Sachverhalt, dass jedes Element der Zielmenge höchstens einmal als Funktionswert angenommen wird, heisst Injektivität.

Folgende Betrachtung zeigt, dass Hashfunktionen nicht streng mathematisch injektiv sind: Jede beliebige Zeichenkette kann Eingabewert für Hashfunktionen sein, es gibt also unendlich viele Eingabewerte. Die Anzahl Ausgabewerte ist jedoch beschränkt, da die Länge des Hash-Werts beschränkt ist. Die Anzahl mögliche Eingabewerte ist also grösser als die Anzahl Ausgabewerte. Es ist daher unmöglich, die Kollisionsfreiheit zu 100% zu garantieren. Da die Anzahl möglicher Ausgabewerte jedoch gross ist, kann man sicherstellen, dass die Kollisionswahrscheinlichkeit gegen null geht.

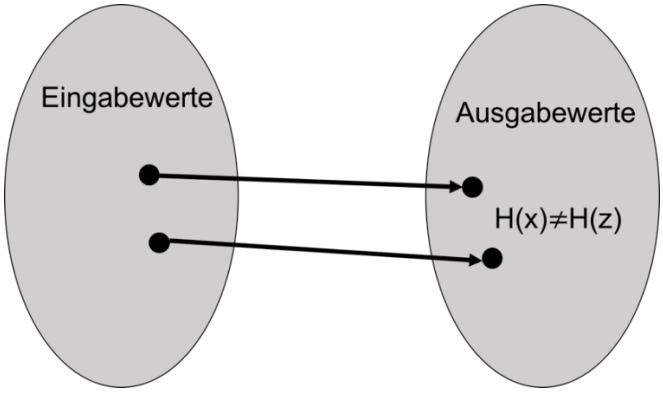

Abbildung 5: Kollisionsfreiheit der Hash-Funktion

Einwegeigenschaft

Hashfunktionen stellen sicher, dass es nicht möglich ist, vom Ausgabewert auf den Eingabewert zu schliessen. Funktionen, welche diese Eigenschaften besitzen, werden Einwegfunktionen genannt. Dies hat zur Folge, dass man von dem kreierten Hashwert nicht auf die ursprünglich eingegebene Zeichenkette schliessen kann.

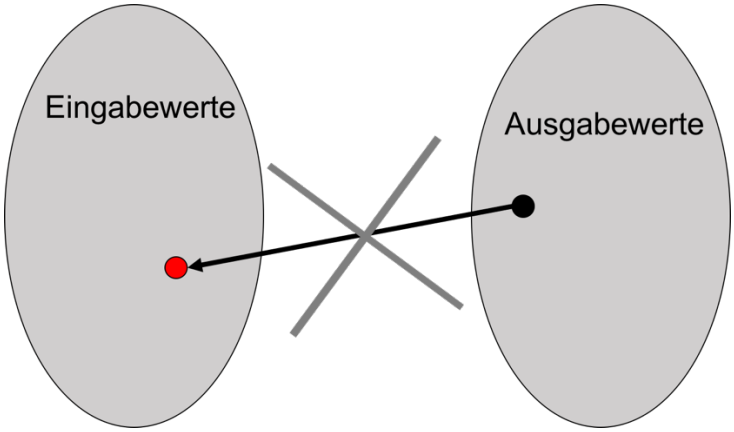

Abbildung 6: Einwegeigenschaft der Hash-Funktion

Determiniertheit

Nach den vorherigen Eigenschaften liegt die Versuchung nahe zu denken, dass die Hashfunktion dem Input einen komplett zufälligen Wert zuordnet. Eine Möglichkeit hierfür wäre zum Beispiel, die aktuelle Zeit mit dem aktuellen Datum und mit den Koordinaten eines Standortes zu verknüpfen. Jedoch sind die Hashfunktionen keine Zufallsfunktionen, sondern besitzen die Eigenschaft der Determiniertheit. Darunter versteht man, dass derselbe Eingabewert stets auf denselben Ausgabewert abgebildet wird. Hashfunktionen sind keine Zufallsfunktionen und stellen trotzdem die Einwegeigenschaft sicher. Ein Hashwert vom Wort „Blockchain“ ist dementsprechend immer gleich, egal, wer die Funktion durchführt, zu welcher Zeit oder an welchem Ort.

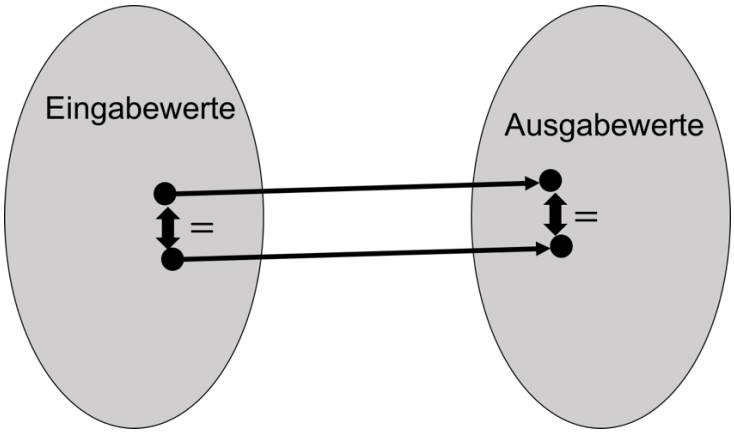

Abbildung 7: Determiniertheit der Hash-Funktion

Pseudozufallsfunktion

Ausserdem ist die Funktion so ausgelegt, dass bei der kleinsten Veränderung des Eingabewertes ein komplett neuer Ausgabewert entsteht. Wenn also bei einem Eingabewert auch nur ein Zeichen geändert wird, entsteht eine komplett neue Zeichenkette als Ausgabewert. Ein anders gesetzter Punkt oder Leerzeichen können dementsprechend einen komplett neuen Hashwert zur Folge haben.

Das Wort „Blockchain“ wird beispielsweise durch den SHA-256 auf den Hashwert 625da44e4eaf58d61cf048d168aa6f5e492dea166d8bb54ec06c30de07db57e1 abgebildet, während das nur leicht veränderte Wort „Blockchain?“ auf den Hashwert 4a48bc5b69ec7587c4672030953978ab2956e32de5fe1ff65a373b5460413c67 abgebildet wird.

Nun wird auch klar, wieso der Hashwert dafür geeignet ist, zwei Informationsstände miteinander zu vergleichen. Jede noch so geringe Veränderung eines Informationsstandes führt zu einem komplett anderen Hashwert. Aufgrund der Kollisionsfreiheit ist es praktisch nicht möglich, dass zwei unterschiedliche Informationsstände denselben Hashwert haben. Zwei identische Informationsstände haben hingegen immer denselben Hashwert, da die Funktion deterministisch ist. [18]

Das Verständnis der Eigenschaften von Hashfunktionen basiert auf vielen interessanten mathematischen Aspekten und wichtigen Inhalten der Informatik. Diese wollen wir im Folgenden verstehen. Um alle Details nachvollziehen zu können, erarbeiten wir erst einige Grundlagen zum Thema Binärsystem und logische Operationen.

2.1.3. Mathematische Grundlagen

Andere Zahlensysteme

Normalerweise bewegen wir uns in der Mathematik beim Darstellen von Zahlen im sogenannten **Dezimalsystem**. Jedoch sind wir uns dessen meist gar nicht aktiv bewusst. Zahlen im Dezimalsystem darzustellen bedeutet, dass wir unsere Zahlen durch Addition von Potenzen der Zahl 10 notieren. Die Zahl 342 zum Beispiel besteht aus $3 \cdot 10^2 + 4 \cdot 10^1 + 2 \cdot 10^0$, die Zahl 1.3 ist aufgebaut durch $1 \cdot 10^0 + 3 \cdot 10^{-1}$.

Für die Informatik ist es häufig von Vorteil das **Binärsystem** zu verwenden. Dieses funktioniert auf dieselbe Art und Weise wie das Dezimalsystem, lediglich mit der Basis 2 statt 10. Die Zahl 14 lässt sich aufteilen in $1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$, wird also als 1110 notiert. Im Binärsystem werden lediglich die Ziffern 0 und 1 verwendet.

Ein weiteres Zahlensystem, welches in der Informatik häufig anzutreffen ist, ist das **Hexadezimalsystem**. Dieses System funktioniert mit der Basis 16. Theoretisch bräuchten wir beim Hexadezimalsystem Ziffern von den Zahlen „1“ bis „15“, jedoch haben wir keine Ziffern für die Zahlen 10, 11, 12, 13, 14, 15. Deswegen weichen wir hier auf Buchstaben aus: A wird für 10 verwendet, B für 11, C für 12, D für 13, E für 14 und F für 15. Die Zahl 31 wird also aufgeteilt in $1 \cdot 16^1 + F \cdot 16^0$ und als 1F notiert. Das Hexadezimalsystem eignet sich ideal zur Darstellung mehrerer Binärwerte (Bit), da zwei Hexadezimalwerte genau 8 Binärwerte darstellen (1 Byte = 8 Bit).

Logische Operationen

Binärzahlen werden oft durch logische Operationen verbunden. Wir werden nun diese Operationen genauer betrachten.

[18] Drescher: Blockchain (2017), S.34-35.

Die Inversion: Bei der Inversion werden die Ziffern einer Binärzahl jeweils invertiert (umgekehrt). Anstelle jeder Null steht neu eine Eins, anstelle jeder Eins, wird eine Null notiert. Diese Operation wird durch das Symbol „¬“ dargestellt.

$\neg 1 = 0$

$\neg 0 = 1$

$\neg(1010011) = 0101100$

Die „oder“-Operation: Die Eingabewerte dieser Operation sind zwei oder mehr Binärzahlen. Es wird jeweils eine Ziffer der einen Zahl mit der Ziffer der entsprechenden Stelle der anderen Zahl verglichen. Wir können uns diese Operation als die Frage „beträgt mindestens eine der Ziffern 1 vorstellen?“ Ist die Antwort auf diese Frage „ja“, so erhalten wir das Resultat 1, ist sie nein, so beträgt unser Resultat „0“. Das Symbol für diese Operation ist ∨.

$1 \vee 1 = 1$

$1 \vee 0 = 1$

$0 \vee 0 = 0$

$100 \vee 010 = 110$

$0 \vee 0 \vee 1 = 1$

Die „und“-Operation: Die Ausgangssituation ist hier analog zur „oder“-Operation. Lediglich die Frage unterscheidet sich, sie lautet hier: „betragen alle Ziffern 1?“ Ist die Antwort auf diese Frage „ja“, so erhalten wir erneut das Resultat 1, ist sie nein, so beträgt unser Resultat „0“. Das Symbol für diese Operation ist ∧.

$1 \wedge 0 = 0$

$1 \wedge 1 = 1$

$110 \wedge 101 = 100$

$1 \wedge 1 \wedge 0 = 0$

Die „exklusives oder“-Operation: Auch hier ist die Ausgangssituation identisch zu den beiden anderen Operationen. Dieses Mal lautet die Frage: „beträgt genau eine der Ziffern 1?“. Als Symbol verwenden wir für diese Operation in diesem Dokument ⊕.

$1 \oplus 0 = 1$

$1 \oplus 1 = 0$

$101 \oplus 011 = 110$

$0 \oplus 1 \oplus 0 = 1$

Die Addition von Binärzahlen

Zwei Binärzahlen werden auf gleiche Art und Weise addiert wie zwei Dezimalzahlen. Von hinten begonnen werden je zwei Ziffern addiert. Überschreitet der dabei erhaltene Wert die Zahl eins, so wird ein Übertrag verwendet, der an der Stelle eins weiter vorne addiert wird.

Genauer:

1+1 = 0 Übertrag 1

1+1+1 = 1 Übertrag 1

```
    1  1  0  1  0
+   1  1  0  1  1
 1  1     1        Überträge
 1  1  0  1  0  1
```

Die Modulo-Addition

Zu Beginn wird bei der Modulo-Addition wie bei der gewöhnlichen Addition zweier Binärzahlen vorgegangen. Im Resultat wird dann nachgesehen, wie oft eine gewisse Zahl (zum Beispiel 2^{32}) darin Platz findet. Exakt so oft wird die Zahl dann abgezogen. Der Rest hiervon ist dann das Endresultat dieser besonderen Addition.

Für diejenigen, welche sich bereits mit der Modulo-Operation auskennen: Formal sieht diese Addition wie folgt aus: $(X + Y) \bmod (2^{32})$.

Ein Beispiel der Modulo Operation wäre 12 mod (4). 12 geteilt durch 4 ergibt genau 3 Rest 0. 12 mod (4) ist also gleich 0. Demnach ist (3 mod (4) gleich 1 und 13 mod (5) gleich 3.

Bei einer Modulo Addition werden zwei Zahlen, beispielsweise 2^{32} und 3^{10}, addiert. Dadurch erhält man 4295026345. Diese Zahl teilt man wiederum durch 2^{32} und erhält Rest 59049. Dieser Rest ist nun das Ergebnis der Modulo Addition.

Die Addition von Hexadezimalzahlen

Hexadezimalzahlen lassen sich auf dieselbe Art und Weise wie Dezimal- und Binärzahlen addieren. Lediglich die Überträge sind schwieriger zu ermitteln, da für uns der Umgang mit den Hexadezimalzahlen recht ungewohnt ist.

A + F ergäbe als Dezimalzahl dargestellt beispielsweise 25. Dies entspricht, als Hexadezimalzahl notiert, 17. 7 wäre hier also das Resultat der bestimmten Ziffer, 1 der Übertrag für die nächsten Ziffern.

Beispiel:

```
    A  F  F  E
+   9  B  2  1
    1  1
= 1  4  B  1  F
```

E + 1 = F

F + 2 ergibt als Dezimalzahl 17, als Hexadezimalzahl 11, 1 Übertrag 1

F + B + 1 ergibt als Dezimalzahl 27, als Hexadezimalzahl 1B, B Übertrag 1

A + 9 + 1 ergibt als Dezimalzahl 20, als Hexadezimalzahl 14, 4 Übertrag 1.[19]

[19] Lerntool.ch: Hexadezimale Addition, heruntergeladen am 12.10.2018.

Buchstaben und Zeichen im Binärsystem

In einem späteren Abschnitt der Arbeit wird es nötig sein, ein Wort in einen Binärcode umzuwandeln. Das hierfür verwendete Verfahren um Buchstaben oder Wörter in einen Binärcode umzuwandeln nennt sich ASCII-Verfahren. Es steht für «American Standard Code for Information Interchange». Es wird jedem Zeichen eine Zahl zugeordnet. Dazu gehören Buchstaben, Zahlen aber auch spezielle Zeichen wie Leerzeichen oder Ausrufezeichen. Diese zugeordneten Zahlen kann man dann als Binärzahl darstellen. Dem Buchstaben „A“ wird beispielsweise der Zahlenwert 65 zugeordnet. Im Binärsystem entspricht dies 1000001. Andere Systeme, welche verwendet werden, um zusätzliche Zeichen als Binärzahl zu schreiben, arbeiten mit 8-bit Zahlen. Damit die Kommunikation mit Computern, welche mit diesen Systemen arbeiten, funktioniert, wird jede Zahl auf 8-bit ergänzt. Dies ist möglich, indem vor der Binärzahl die benötigte Anzahl an Nullen eingefügt werden. Also wird aus dem Buchstaben A die Binärzahl 01000001. Wenden wir dieses System nun auf das Wort Baden an: Baden besteht aus fünf Buchstaben. Mit der ASCII-Tabelle codiert führen diese Buchstaben auf die fünf Zahlen 66, 97, 100, 101 und 110. Diese fünf Dezimalzahlen wandeln wir als nächstes in fünf Binärzahlen um. Diese lauten 01000010, 01100001, 01100100, 01100101 und 01101110. Baden wird dementsprechend als 0100001001100001011001000110010101101110 geschrieben.[20]

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Abbildung 8: ASCII Tabelle. Wikipedia, 09.10.2018

2.1.4. Wie funktioniert der SHA-256?

Die Hashfunktion, welche am häufigsten in der Blockchain verwendet wird, ist der SHA-256. Wir werden nun an Hand des SHA-256 verstehen, wie eine Hashfunktion genau aufgebaut ist.

Die Entstehung der Wörter

Als Eingabewert wählen wir uns eine Zeichenkette aus. Wir nehmen in diesem Beispiel das Wort «Baden». Dieses Wort wird jetzt in den Binärcode übersetzt. Das Ergebnis hiervon ist die Abfolge «01000010 01100001 01100100 01100101 01101110». Hinter diesem Binärcode wird noch eine Eins angehängt. Nun wird dies in Zeilen mit einer Länge von jeweils 32 Binärwerten (Bits) eingetragen. Es werden so viele Nullen aufgefüllt, dass 16 Zeilen mit je 32 Binärwerten entstehen. Die letzten Zeichen werden so abgeändert, dass sie die Länge der ur-

[20] The Tech Train: ASCII, heruntergeladen am 25.10.2018

sprünglichen Nachricht in Bits „gemessen“ (in diesem Fall 40) angeben. Es entsteht eine Gesamtlänge von 512 Bits. Die 32 Bits langen Zeilen werden „Wörter“ genannt. Der Index eines Wortes beschreibt, um das „wievielte“ Wort es sich handelt:

1. 01000010011000010110010001100101 = W_1
2. 01101110100000000000000000000000 = W_2
3. 00000000000000000000000000000000 = W_3
4. 00000000000000000000000000000000 = W_4
5. 00000000000000000000000000000000 = W_5
6. 00000000000000000000000000000000
7. 00000000000000000000000000000000
8. 00000000000000000000000000000000
9. 00000000000000000000000000000000
10. 00000000000000000000000000000000
11. 00000000000000000000000000000000
12. 00000000000000000000000000000000
13. 00000000000000000000000000000000
14. 00000000000000000000000000000000
15. 00000000000000000000000000000000
16. 00000000000000000000000000101000

Es werden weitere 48 Wörter angehängt.

Dabei entsteht ein Wort durch eine Modulo-Addition. Wie bei der Erklärung zur Modulo-Addition wird die Zahl 2^{32} zur Moduloberechnung verwendet. Der Einfachheit halber ist die Modulo-Addition jeweils durch ein gewöhnliches „+“-Zeichen dargestellt. Das Wort mit dem Index zwei tiefer wird in eine Funktion eingesetzt und zu dem Wort mit dem um sieben tieferen Index addiert. Dann wird das Wort mit dem um 15 geringeren Index in eine andere Funktion eingesetzt und addiert. Schlussendlich wird das Wort mit dem um 16 geringeren Index hinzugefügt.

In einer Formel beschrieben sieht dies folgendermassen aus:

$$W_i = \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16}$$

Hierbei werden zwei Funktionen verwendet σ_1 und σ_0, welche wir nun genauer beschreiben.

Die Funktionen selbst bestehen wiederum aus verschiedenen Basisfunktionen, welche miteinander verknüpft werden. Die erste dieser verwendeten Basisfunktionen heisst „Rotate Right“. Rotate Right nimmt als Input ein „Wort“ und eine natürliche Zahl. Alle Ziffern des Wortes werden um die natürliche Zahl nach rechts verschoben. Die Stellen, die rechts wegfallen, werden links wieder angehängt, eine Art Rotation also. Die Funktion wird abgekürzt durch RotR. Dabei steht das RotR(W_1,7) dafür, dass man beim ersten Wort jede Ziffer um sieben Stellen nach rechts verschiebt. Alle Stellen welche rechts «wegfallen» werden links wieder angehängt. Demnach wird unser 1. Wort «01000010011000010110010001100101» zu «11001010100001001100001011001000».

Die nächste Basisfunktion nennt sich „Shift Right“. Auch sie nimmt als Eingabewert ein „Wort“ und eine natürliche Zahl. Erneut werden alle Ziffern des Wortes um eine natürliche Zahl nach rechts verschoben. Bei dieser Funktion fallen jedoch die Ziffern rechts ganz weg und links wird mit Nullen ergänzt.

Bei der Funktion ShR(W_1,3) verschiebt man jede Ziffer des ersten Wortes um 3 Stellen nach rechts, jedoch werden alle rechts wegfallenden Stellen einfach links mit Nullen ergänzt. Aus

unserem 1. Wort wird also nach Ausführen der Basisfunktion «00001000010011000010110010001100».

Die Operation, die zum Verknüpfen einzelner Werte verwendet wird, heisst „XOR“ oder „Exklusives Oder“. Wir erinnern uns: Die Operation entspricht dem Ausdruck: entweder oder (aber nicht beides). Hierbei werden zwei Stellen einer Binärzahl verglichen. Beträgt einer der beiden (also entweder oder) eine Eins, so ist das Resultat Eins. Betragen beide Stellen Eins oder Null, so ist das Resultat Null.

Nun sind wir dazu bereit, die beiden oben beschriebenen Funktionen genau zu verstehen. Formal ausgedrückt sehen sie folgendermassen aus:

$$\sigma_0(X) = RotR(X,7) \oplus RotR(X,18) \oplus ShR(X,3)$$

$$\sigma_1(X) = RotR(X,17) \oplus RotR(X,19) \oplus ShR(X,10)$$

Bei der ersten Funktion wird das eingegebene Wort X einmal um 7 nach rechts rotiert, einmal um 18 nach rechts rotiert und einmal um 3 nach rechts „geshiftet“. Diese 3 Ausdrücke werden dann durch ein „exklusives oder“ miteinander verbunden.

Bei der zweiten Funktion wird das eingegebene Wort X einmal um 17 nach rechts rotiert, einmal um 19 nach rechts rotiert und einmal um 10 nach rechts „geshiftet“. Diese 3 Ausdrücke werden dann erneut durch ein „exklusives oder“ miteinander verbunden.

Wir verwenden nun unser Eingangsbeispiel und führen den ganzen Prozess an Hand dieses Beispiels durch.

Will man das Wort W_{17} ausrechnen, sucht man zuerst die Wörter W_{15}, W_{10}, W_2 und W_1. Da W_{15} und W_{10} lediglich aus Nullen bestehen kann man diese zwei Summanden weglassen. Eine Addition mit Null ändert schliesslich nichts, auch an der darauffolgenden Moduloberechnung.

RotR(W_2,7) = 00000000110111010000000000000000

RotR(W_2,18) = 00000000000000000011101110100000

ShR(W_2,3) = 00001101110100000000000000000000

Wir wenden nun σ_0 auf W_2 an.

$RotR(W_2,7) \oplus RotR(W_2,18) \oplus ShR(W_2,3)$ = 00001101000011010001101110100000

Diese Binärzahl addieren wir nun zu W_1.

00001101000011010001101110100000

\+ 01000010011000010110010001100101

= 01001111011011101000000000000101

Da 2^{32} um einiges grösser ist als das Resultat unserer Addition, muss 2^{32} nicht abgezogen werden und der Rest ist exakt die Zahl, die wir durch die Addition erhalten haben.

W_{17} = 01001111011011101000000000000101

Das Abziehen von 2^{32} ist nur dann notwendig, wenn die Summe grösser als 2^{32} ist. In diesem Fall dient die Modulo-Operation dazu, wieder eine 32-Stellige Binäre Zahl zu erhalten. Dieses Vorgehen führt man nun solange durch, bis man beim 64. Wort angelangt ist.

Zusammenfassen der Wörter

Für den nächsten Schritt definieren wir verschiedene Konstanten. Dies sind 64 Konstanten mit Namen „K“ und 8 Konstanten mit Namen „H“. Das K_1 wird wie folgt generiert.

Die ersten 32 Nachkommastellen von $\sqrt[3]{2} \cong 1.25992101496991873$ multiplizieren wir mit 16^8. $0.25992101496991873 * 16^8 = 1116352258$. Das Resultat der Multiplikation notieren wir als Hexadezimalzahl, in unserem Falle 0x428a2f98.

Die restlichen Konstanten K werden mit dem gleichen Prinzip berechnet. Lediglich die Zahl unter der Wurzel ändert sich. Anstatt der 2 werden die dritten Wurzeln der darauffolgenden Primzahlen berechnet. Also für die zweite Konstante 3, dann 5, dann 7, dann 11 und so weiter. Mit den Konstanten H wird gleich vorgegangen. Statt der dritten Wurzel wird hier die zweite Wurzel gezogen.

In diesem Schritt benutzen wir vier neue Funktionen.

$$e_0 = RotR(X,2) \oplus RotR(X,13) \oplus RotR(X,22)$$

$$e_1 = RotR(X,6) \oplus RotR(X,11) \oplus RotR(X,25)$$

Diese zwei Funktionen sind sehr ähnlich aufgebaut wie diejenigen, welche wir bei der Erstellung der Wörter verwendet haben. Wir überlassen es dem interessierten Leser, diese selber an einer beliebigen Binärzahl durchzuführen.

Die nächsten zwei Funktionen benutzen die logische Operationen, welche wir weiter oben erläutert haben.

$$Ch(X,Y,Z) = (X \wedge Y) \oplus (\neg X \wedge Z)$$

$$Maj(X,Y,Z) = (X \wedge Y) \oplus (X \wedge Z) \oplus (X \wedge Z)$$

Wir führen nun Variablen ein, welche uns beim Zusammenfügen aller Wörter helfen werden.

a = H_1

b = H_2

c = H_3

d = H_4

e = H_5

f = H_6

g = H_7

h = H_8

Nun wird die folgende Abfolge von Rechnungen 64-mal durchgeführt. Das i läuft hierbei von 1 bis 64. Bei dem Additionszeichen handelt es sich wie zuvor um eine Modulo-Addition.

T1 = h + e_1(e) + Ch(e,f,g) + K_i + W_i

T2 = e_o(a) + Maj(a,b,c)

h = g

g = f

f = e

e = d + T1

d = c

c = b

b = a

a = T1 + T2

Das W bezieht sich auf die Wörter, welche wir im vorherigen Schritt definiert haben. Am Schluss jeder Runde hat dementsprechend jede Variable einen neuen Wert. Bei der nächsten Runde wird dann dieser Wert verwendet.

Genau diese Rechnungen führen wir nun mit unseren Wörtern durch. Die verschiedenen Variablen sehen nach der ersten Runde wie folgt aus:

a = 3E69ECB2

b = 6a09e667

c = bb67ae85

d = 3c6ef372

e = DB294707

f = 510e527f

g = 9b05688c

h = 1f83d9ab

Nach der zweiten Runde:

a = 439EFBE5

b = 3E69ECB2

c = 6a09e667

d = bb67ae85

e = F7EAC016

f = DB294707

g = 510e527f

h = 9b05688c

Und schlussendlich nach der 64. Runde:

a = AE5E947C

b = 4C4518B3

c = D6C4FBF5

d = 69B4FD6F

e = 0B79F779

f = 664B2266

g = E700EC75

h = 47BF1640

Zuletzt wird das a, welches nach der 64. Runde entsteht, zu H_1 addiert (erneut durch Modulo-Addition) und ergibt unseren gewünschten ersten Teil des Schlussresultats. Das b wird dann zu H_2 addiert, das c zu H_3 und so weiter. Wenn man nun alle Resultate dieser Additionen hintereinander hängt, erhält man den Hashwert unseres ursprünglichen Wortes «Baden».

Wir zeigen dies nun an Hand von unserem a und unserem H_1.

$a + H_1$

$= AE5E947C + H_1$

$= 18687AE3$

Dies wird nun analog mit b, c, d, e, f, g, h und den weiteren Konstanten durchgeführt. Schlussendlich erhält man den Hashwert von Baden, welcher wie folgt lautet:

18687AE3 07ACC738 1333EF67 0F04F2A9 5C8849F8 01508AF2 0684C620 A39FE359

Unsere Nachricht hat eine Länge, welche geringer als 512 Bits ist, weshalb sie sich anfangs problemlos in 16 Wörter aufteilen liess. Wäre eine Nachricht jedoch länger als 512 Bits, müsste auf das nächste Vielfache von 512 ergänzt werden, also beispielsweise auf 2 Blöcke von 16 Wörtern aus 32 Zeichen, was eine Länge von 1024 Bits ergibt.

Nun würden wir auf den ersten dieser Blöcke separat alle Operationen so ausführen, wie wir es in den vorherigen Schritten getan haben.

Anstatt nun aufzuhören, indem wir $a + H_1$ als den ersten Teil unseres Hashwerts abspeichern, ändern wir a in $a + H_1$, b in $b + H_2$ uns so weiter. Nun führen wir alle Rechnungen mit den Wörtern zweiten Block erneut 64-mal durch. Das neue $a + H_1$ addieren wir nun zu dem vorherigen. Bei zusätzlichen Blöcken muss so oft auf diese Weise vorgegangen werden, bis alle Blöcke im Hashwert mit einbezogen sind.[21]

2.1.5. Wahrscheinlichkeitsberechnungen zur Einwegeigenschaft

Es ist nicht ausgeschlossen, dass man zu einem gegebenen Hashwert einen Eingabewert findet, welcher exakt diesen Hashwert hat. In diesem Abschnitt werden wir berechnen, wie gross die Wahrscheinlichkeit ist, dies zu erreichen. Um den Eingabewert zu finden müsste man einen Computer die Hashwerte von sehr vielen verschiedenen Eingabewerten durchrechnen lassen. Beispielsweise könnte man mit dem Wort „Hallo“ beginnen und immer mehr Leerzeichen hinten anhängen, bis der gewünschte Hashwert erreicht ist. Vereinfacht gehen wir hierbei davon aus, dass Hashwerte gleichverteilt sind. Dies bedeutet, dass alle Hashwerte gleich häufig vorkommen.

Wir stellen uns eine Hashfunktion vor, welche einer Zeichenkette eine dreistellige Binärzahl zuordnet. Wie gross wäre dann die Wahrscheinlichkeit einen Eingabewert für den Hashwert 101 zu erraten? Begonnen wird mit einem zufälligen Eingabewert. Im binären Zahlensystem gibt es für jede Stelle des Hashwerts lediglich zwei Möglichkeiten, eine 0 oder eine 1. Die Wahrscheinlichkeit, dass der Hashwert zum Eingabewert an der ersten Stelle eine 1 hat, ist folglich $\frac{1}{2}$. Nun gibt es für beide Möglichkeiten der ersten Ziffer jeweils zwei Möglichkeit, welche die zweite Ziffer annehmen könnte. Die Wahrscheinlichkeit, dass beim Hashwert vom geratenen Eingabewert die ersten zwei Ziffern korrekt sind, liegt bei $\frac{1}{2} * \frac{1}{2} = \frac{1}{2^2} = \frac{1}{4}$. Für die dritte Ziffer gibt es nun für die vier bereits bestehenden Möglichkeiten wieder jeweils zwei Möglichkeiten also $\frac{1}{2} * \frac{1}{2} * \frac{1}{2} = \frac{1}{2^3} = \frac{1}{8}$. Die Chance einen Eingabewert zu finden, welcher die Zahl 101 als Hashwert hat, liegt somit bei $\frac{1}{2^3} = \frac{1}{8}$. Die Zahl im Nenner besteht aus einer Zweierpotenz. Die Basis beschreibt die Anzahl der Zeichen, die möglich sind und der Exponent die

[21] The cryptographic hash function SHA-256, https://www.researchgate.net/file.PostFileLoader.html?id=534b393ad3df3e04508b45ad&assetKey=AS%3A273514844622849%401442222429260, heruntergeladen am 22.08.2018.

Länge der Zeichenkette. Die Wahrscheinlichkeit, dass der Eingabewert nicht den gewünschten Hashwert hat ist:

$$1 - \frac{1}{2^3} = \frac{7}{8} = 0.875$$

Falls der gewünschte Hashwert nicht erreicht wird, würde ein neuer Versuch mit einem anderen Eingabewert durchgeführt. Die Wahrscheinlichkeit bei zwei unabhängigen Versuchen den gewünschten Hashwert nicht zu finden liegt bei:

$$(1 - \frac{1}{2^3})^2 = (\frac{7}{8})^2 = \frac{49}{64} \sim 0.77$$

Die Wahrscheinlichkeit das Ziel zu verfehlen nimmt demnach ab, je mehr Eingabewerte ausprobiert werden. Wir wollen nun berechnen, wie wahrscheinlich es ist, dass der gewünschte Hashwert erreicht wird. Hierfür subtrahieren wir die oben berechnete Wahrscheinlichkeit von 1. Die Wahrscheinlichkeit, dass der gewünschte Hashwert mit zwei Versuchen mindestens einmal erreicht wird, liegt demnach bei:

$$1 - \left(1 - \frac{1}{2^3}\right)^2 = 1 - \frac{49}{64} = 0.23$$

Damit etwas als wahrscheinlich gilt, muss die Wahrscheinlichkeit, dass dieser Zustand zutrifft, bei mindestens $\frac{1}{2}$ liegen. Wir wollen jetzt wissen, wie viele Versuche man bräuchte, dass es wahrscheinlich ist, einen Eingabewert zum gesuchten Hashwert zu finden. Folglich muss gelten:

$$1 - \left(1 - \frac{1}{2^3}\right)^x \geq \frac{1}{2}$$
$$1 - \frac{1}{2} \geq \left(1 - \frac{1}{2^3}\right)^x$$

Nun wird der natürliche Logarithmus von beiden Seiten berechnet. Hierbei wird das Logarithmusgesetzt verwendet, welches besagt, dass man einen Exponenten (wie gezeigt) aus einem Logarithmus ausklammern kann:

$$\ln(1 - \frac{1}{2}) \geq x * \ln(1 - \frac{1}{2^3})$$

Als nächstes wird durch den $\ln(1 - \frac{1}{2^3})$ dividiert. $1 - \frac{1}{2^3}$ liegt zwischen 0 und 1, der ln davon ist dementsprechend negativ. Deswegen kehrt sich bei der Division das Relationszeichen um:

$$\frac{\ln(1 - \frac{1}{2})}{\ln(1 - \frac{1}{2^3})} \leq x$$

Da $\frac{\ln(1-\frac{1}{2})}{\ln(1-\frac{1}{2^3})} \sim 5{,}19$ ist, muss man mindestens 6 unabhängige Versuche durchführen, damit die Wahrscheinlichkeit, den gewünschten Eingabewert zu finden, bei 50% oder höher liegt.

Das Vorgehen von oben verallgemeinern wir in einem nächsten Schritt. Definieren wir den Buchstaben N als die Anzahl an möglichen Kombinationen für den Hashwert. In unserem vorigen Beispiel wären dies 2^3. N muss sicherlich positiv sein. Der Buchstabe x entspricht erneut der Anzahl an Versuchen, die man durchführen muss, um den gesuchten Eingabewert wahrscheinlich zu finden.

Analog zum Beispiel muss nun gelten:

$$1-\left(1-\frac{1}{N}\right)^{x} \geq \frac{1}{2}$$
$$1-\frac{1}{2} \geq \left(1-\frac{1}{N}\right)^{x}$$
$$\ln(1-\frac{1}{2}) \geq x * \ln(1-\frac{1}{N})$$
$$(1): \frac{\ln(1-\frac{1}{2})}{\ln(1-\frac{1}{N})} \leq x$$

Für ein sehr grosses N ist der Nenner des Bruches sehr nahe bei Null. Ein gewöhnlicher Rechner ist deswegen nicht im Stande, das x in dieser Formel für ein grosses N zu berechnen. Deshalb suchen wir eine weitere Möglichkeit für eine Formel, welche erneut berechnet, wann die Wahrscheinlichkeit, einen passenden Eingabewert zu finden, sicher grösser als 50% ist. Ist die Bedingung (2) erfüllt, so ist die oben genannte Bedingung (1) sicherlich erfüllt. Wir zeigen also, dass (2)⇒(1):

$$(2): x \geq N * |\ln\left(1-\frac{1}{2}\right)|$$

Für ein reelles x gilt: $(1-x) \leq e^{-x}$. Dies kann nachvollzogen werden, indem man die Graphen der beiden Funktionen 1-x bzw e^{-x}ansieht. e^{-x} befindet sich immer oberhalb von $(1-x)$, bei x = 0 berühren sich die beiden Graphen.

Daher gilt, indem man x durch $\frac{1}{N}$ ersetzt, $\ln(1-\frac{1}{N}) \leq -\frac{1}{N}$.

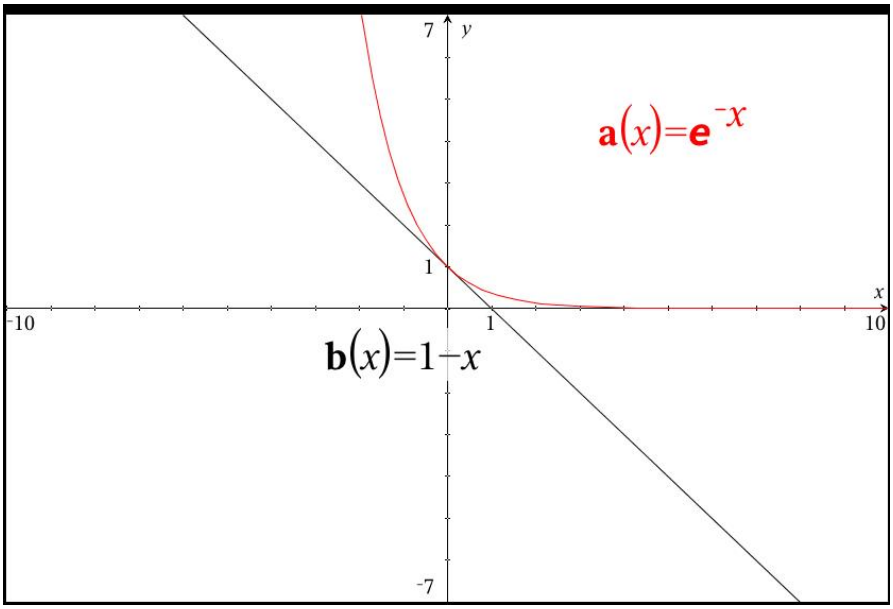

Abbildung 9: Graphen der Funktionen 1-x und e^{-x}

Sowohl $-\frac{1}{N}$ als auch $\ln(1-\frac{1}{N})$ sind negativ. Deswegen können wir beidseitig den Betrag nehmen und das Relationszeichen umdrehen:

$$|\ln(1-\frac{1}{N})| \geq \frac{1}{N}$$

Bildet man beidseitig die Kehrwerte und kehrt erneut die Vorzeichen, so erhält man:

$$\frac{1}{|\ln(1-\frac{1}{N})|} \leq N$$

Auf Grund der Transitivität der „kleiner-Relation“ können wir in

(2) für N den Ausdruck $\frac{1}{|ln\,(1-\frac{1}{N})|}$einsetzen. Neu wird aus (°) deshalb:

$$x \geq \frac{|\ln\left(1-\frac{1}{2}\right)|}{|ln\,(1-\frac{1}{N})|}$$

Beide Logarithmen sind negativ, ein negativer Ausdruck geteilt durch einen negativen ergibt erneut einen positiven Ausdruck. Deswegen können wir die Betragsstriche weglassen und erhalten somit (1). Daraus schliessen wir, dass (1) sicher gilt, falls (2) gilt.

Wir können nun (2) für die SHA-256 Funktion anwenden:

$$x \geq 2^{256} * |\ln\left(1-\frac{1}{2}\right)|$$

Dies ergibt ungefähr $8 * 10^{76}$. Klaus Pommerening beschreibt in seinem Dokument: «Die Sicherheit kryptographischer Verfahren», dass ein Rechner (2 GHz) $6.4 * 10^{16}$ CPU-Zyklen pro Jahr leistet. Diese Leistung entspricht etwa der eines durchschnittlichen Laptops. Es würde demnach im Schnitt $\frac{8*10^{76}}{6.4*10^{16}} = 1.3 * 10^{60}$ Jahre dauern, um von einem Hash auf die ursprüngliche Zeichenkette zu schliessen.

2.1.6. Wahrscheinlichkeitsberechnung zur Kollisionsfreiheit

Wir wollen uns damit befassen, wie wahrscheinlich es ist, dass zwei verschiedene Inputs auf den gleichen Output führen, also kollidieren. Um dies zu bewältigen, sehen wir uns das sogenannte Geburtstagsparadoxon an. Beim Geburtstagsparadoxon wird folgende Frage gestellt: «Wie viele Menschen müssen sich in einem Raum befinden, dass die Wahrscheinlichkeit, dass zwei am selben Tag Geburtstag haben, mindestens 50% ist». Der Grund, weshalb dies ein Paradoxon genannt wird, ist, dass ein Grossteil der Menschheit diese Frage intuitiv ganz falsch beantwortet. Die richtige Antwort ist nämlich 23, wobei meist eine viel höhere Zahl erwartet wird. Um auf diese Zahl zu stossen, braucht es ein wenig Wahrscheinlichkeitsrechnung.

Nehmen wir, an es gäbe genau 365 Tage im Jahr und es ist an jedem Tag gleich wahrscheinlich, Geburtstag zu haben. Wir wollen zunächst wissen, mit welcher Wahrscheinlichkeit die Personen im Raum alle an einem anderen Tag Geburtstag haben. Bei einer Person ist es klar, nämlich 1. Dies kann man aber auch als $\frac{365}{365}$ schreiben. Die zweite Person darf nicht mehr an diesem Tag Geburtstag haben, es besteht hierzu noch eine Wahrscheinlichkeit von $\frac{364}{365}$. Bei der dritten gäbe es dann noch eine Wahrscheinlichkeit von $\frac{363}{365}$. Um die Gesamtwahrscheinlichkeit zu berechnen, muss man diese Wahrscheinlichkeiten nun multiplizieren: $1 * \frac{364}{365} * \frac{363}{365} \sim 0{,}992$. Somit liegt die Wahrscheinlichkeit, dass von drei zufälligen Personen keine zwei Personen am gleichen Tag Geburtstag haben, bei ungefähr 99,2%.

Die Wahrscheinlichkeit, dass in diesem Fall zwei Personen am selben Tag Geburtstag haben ist dementsprechend $1 - 0{,}992 = 0{,}008$.

Befinden sich r Personen in einem Raum, so wird die Wahrscheinlichkeit, dass zwei Personen am selben Tag Geburtstag haben, durch folgende Formel berechnet: $P = 1 - \left(\frac{N}{N} * \frac{N-1}{N} * \frac{N-2}{N} * \ldots * \frac{N-r+1}{N}\right) = 1 - \left(\left[1 - \frac{1}{N}\right] * \left[1 - \frac{2}{N}\right] * \ldots * \left[1 - \frac{r-1}{N}\right]\right)$. „r“ beschreibt hier die Anzahl Personen und „N“ die Anzahl an Tagen. Allgemein kann „r“ als die Anzahl Versuche beschrieben werden, während „N“ die Anzahl an Ausprägungen beschreibt. Dabei berechnet man eigentlich die Wahrscheinlichkeit, dass nie dieselbe Ausprägung angenommen wird, subtrahiert diese dann von 1 und erhält damit die Wahrscheinlichkeit, dass eine Kollision stattgefunden hat.

Gehen wir nochmals auf unser Geburtstagsbeispiel zurück: In dieser Formel beschreibt die Wahrscheinlichkeit P, wie gross die Wahrscheinlichkeit für eine Kollision von Geburtstagen ist. Berechnet man die Wahrscheinlichkeit mit dieser Formel für die 365 Tage im Jahr und 23 Personen, so erhält man eine ungefähre Kollisionswahrscheinlichkeit von 50%.

Die obige Formel hat den Nachteil, dass man die vielen Terme in der Klammer einzeln berechnen und multiplizieren muss. Dies dauert sehr lange für ein grosses r, weshalb wir einen anderen Weg finden müssen. Wir können Werte festlegen, welche sicherlich grösser oder kleiner als unser P sind. Wenn wir diese Werte gefunden haben, wissen wir, dass unser P dazwischenliegen muss. Wir nennen diese Werte obere und untere Grenze.

In einem ersten Schritt werden wir zeigen, dass eine obere Grenze bei

$\frac{0}{N} + \frac{1}{N} + \frac{2}{N} + \cdots + \frac{r-1}{N}$ liegt.

Es ist klar, dass es sinnlos wäre, diese Rechnung für mehr als N Versuche durchzuführen. Wenn wir mehr Versuche tätigen, als es Möglichkeiten gibt, ist schliesslich beinahe sicher, dass eine Kollision entsteht. Es sei also $1 \leq r \leq N$.

Sehen wir uns erneut den Klammerterm $\left(\left[1 - \frac{0}{N}\right] * \left[1 - \frac{1}{N}\right] * \left[1 - \frac{2}{N}\right] * \ldots * \left[1 - \frac{r-1}{N}\right]\right)$ an. Diesen kann man ausmultiplizieren. Dafür muss man bekanntlich jeden Wert in einer Klammer mit jedem anderen Wert in allen anderen Klammern multiplizieren. Am Schluss werden dann alle Werte addiert. Wir fangen zuerst damit an, alle Einsen miteinander zu multiplizieren. Der erste Summand ist folglich $1^r = 1$.

Als nächstes lässt sich immer eine Klammer auswählen. Aus dieser ausgewählten Klammer wird der Bruch mit den Einsen aus allen anderen Klammern multipliziert. Hierbei kann jede Klammer genau einmal ausgewählt werden, sodass r Summanden entstehen:

$1^{r-1} * \left(-\frac{0}{N}\right) + 1^{r-1} * \left(-\frac{1}{N}\right) + \cdots + 1^{r-1} * \left(-\frac{r-1}{N}\right)$

Der erste Summand ist 1 und die nächsten r Summanden sind $-\frac{0}{N} - \frac{1}{N} - \cdots - \frac{r-1}{N}$.

Als nächstes wählen wir zwei Klammern aus, aus denen wir die Brüche multiplizieren, von allen anderen wird erneut mit der Eins multipliziert. Ab diesem Punkt lassen wir die jeweiligen Summanden mit 0 im Zähler weg, da diese dem Wert 0 entsprechen und bei einer Addition somit nichts verändern. Es entsteht also ein Term folgender Art:

$1^{r-2} * \frac{1*2}{N^2} + 1^{r-2} * \frac{1*3}{N^2} + \cdots + 1^{r-2} * \frac{(r-2)(r-1)}{N^2} = \frac{1*2}{N^2} + \frac{1*3}{N^2} + \cdots + \frac{(r-2)(r-1)}{N^2}$

Analog für die nächsten Summanden:

$1^{r-3} * \left(-\frac{1*2*3}{N^3}\right) + 1^{r-3} * \left(-\frac{1*2*4}{N^3}\right) + \cdots + 1^{r-3} * \left(-\frac{(r-3)(r-2)(r-1)}{N^3}\right) = -\frac{1*2*3}{N^3} - \frac{1*2*4}{N^3} - \cdots - \frac{(r-3)(r-2)(r-1)}{N^3}$

Dies führen wir fort, bis alle Möglichkeiten an Multiplikationen durchgeführt sind. Dazu müssen wir glücklicherweise nicht alles ausmultiplizieren. Den Grund dafür werden wir gleich verstehen. Alle diese Summanden werden schlussendlich von 1 subtrahiert:

$$1-\left(1-\frac{0}{N}-\frac{1}{N}-\cdots-\frac{r-1}{N}\ +\frac{1*2}{N^2}+\frac{1*3}{N^2}+\cdots+\frac{(r-2)(r-1)}{N^2}\ -\frac{1*2*3}{N^3}-\frac{1*2*4}{N^3}-\cdots-\frac{(r-3)(r-2)(r-1)}{N^3}+\cdots-\cdots+\cdots-\cdots\right)$$

Die erste 1 fällt weg und bei den anderen Summanden werden jeweils die Vorzeichen umgekehrt, da zum Beispiel $1-(1-\frac{1}{N})=\frac{1}{N}$ und $1-(1-\frac{r-1}{N})=\frac{r-1}{N}$ ist. Es entsteht also der Term:

$$\frac{1}{N}+\frac{2}{N}+\cdots+\frac{r-1}{N}\ -\frac{1*2}{N^2}-\frac{1*3}{N^2}-\cdots-\frac{(r-2)(r-1)}{N^2}\ +\frac{1*2*3}{N^3}+\frac{1*2*4}{N^3}+\cdots+\frac{(r-3)(r-2)(r-1)}{N^3}-\cdots+\cdots-\cdots$$

Wir betrachten nun die folgende Teilsumme:

$$-\frac{1*2}{N^2}-\frac{1*3}{N^2}-\cdots-\frac{(r-2)(r-1)}{N^2}\ +\frac{1*2*3}{N^3}+\frac{1*2*4}{N^3}+\cdots+\frac{(r-3)(r-2)(r-1)}{N^3}$$

Wir wollen jetzt die negativen Terme mit denen positiven Termen vergleichen. Dazu können wir die negativen Terme mit N erweitern, damit wir denselben Nenner haben wie bei den positiven Termen. Sie stellen sich dann wie folgt dar:

$$-\frac{\left(1*2+1*2*3+\cdots+(r-2)(r-1)\right)*N}{N^3}$$

und

$$\frac{\left(1*2*3+1*2*4+\cdots+(r-3)(r-2)(r-1)\right)}{N^3}$$

Der obere Term ist betragsmässig sicherlich grösser als der untere, da der Nenner gleich ist aber im Zähler mit der grössten möglichen Zahl, also N, multipliziert wird. Aufgrund dieser Gegebenheit kann man sagen, dass:

$$-\frac{\left(1*2+1*2*3+\cdots+(r-2)(r-1)\right)*N}{N^3}+\frac{\left(1*2*3+1*2*4+\cdots+(r-3)(r-2)(r-1)\right)}{N^3}\leq 0$$

Beim Konstruieren des Gesamtterms haben wir bereits gesehen, dass die Termgruppen abwechselnd positiv und negativ sind. Wir können somit auch auf alle nachfolgenden Paare von Teilsummen die oben beschriebene Taktik anwenden und zeigen, dass die Summe der Paare stets negativ ist. Es wird dazu immer die negative Teilsumme mit N erweitert. Somit ist der Zähler der negativen Teilsumme grösser als der der positiven Teilsumme.

Wir weisen nun den Betrag der Summen der oben beschriebenen Teilsummen einer Variablen y zu. Damit können wir schreiben

$$P=\frac{0}{N}+\frac{1}{N}+\frac{2}{N}+\cdots+\frac{r-1}{N}-y$$

Da y>0 sieht man damit sofort, dass

$P\leq\frac{0}{N}+\frac{1}{N}+\frac{2}{N}+\cdots+\frac{r-1}{N}$. Wir haben damit gezeigt, dass $\frac{0}{N}+\frac{1}{N}+\frac{2}{N}+\cdots+\frac{r-1}{N}$ eine obere Grenze für P ist.

Wir klammern nun aus der obigen Summe $\frac{1}{N}$ aus, wenden die Gausssche Summenformel auf den Term $0+1+\cdots+r-1$ an und erhalten $\frac{r(r-1)}{2}$. Die obige Summe kann damit dargestellt weren als $\frac{r(r-1)}{2N}=\frac{r^2}{2N}-\frac{r}{2N}$. Wir können also schreiben $P\leq\frac{r^2}{2N}-\frac{r}{2N}$.

In einem zweiten Schritt bestimmen wir nun eine untere Grenze für P. Wir wollen wissen, welchen Wert P sicherlich nicht unterschreiten kann.

Die untere Grenze basiert auf der Tatsache, dass $1 - x \leq e^{-x}$. Diesen Sachverhalt haben wir oben mit Hilfe der Graphen der Terme gezeigt.

Wir setzen $x = \frac{k}{N}$ und erhalten $1 - \frac{k}{N} \leq e^{-\frac{k}{N}}$.

Dies können wir nun für alle k=0, 1, 2, ...r-1 in der oben abgeleiteten Formel für 1-P

$$\text{1-P}=\left(\left[1-\frac{0}{N}\right]*\left[1-\frac{1}{N}\right]*\left[1-\frac{2}{N}\right]*\ldots*\left[1-\frac{r-1}{N}\right]\right)$$

anwenden. Wir erhalten, da $1 - \frac{0}{N} \leq e^{-\frac{0}{N}} = 1$, den Ausdruck

$$\left(\left[e^{-\frac{1}{N}}\right]*\left[e^{-\frac{2}{N}}\right]*\ldots*\left[e^{-\frac{r-1}{N}}\right]\right)$$

Damit gilt

$$1-P=\left(\left[1-\frac{1}{N}\right]*\left[1-\frac{2}{N}\right]*\ldots*\left[1-\frac{r-1}{N}\right]\right)\leq\left(\left[e^{-\frac{1}{N}}\right]*\left[e^{-\frac{2}{N}}\right]*\ldots*\left[e^{-\frac{r-1}{N}}\right]\right)$$

Durch einfache Umformung sehen wir, dass

$$P\geq 1-\left(\left[e^{-\frac{1}{N}}\right]*\left[e^{-\frac{2}{N}}\right]*\ldots*\left[e^{-\frac{r-1}{N}}\right]\right)$$

Durch Anwendung des Potenzgesetzes : $a^b * a^c = a^{b+c}$ und der Gaussschen Summenformel können wir den Klammerausdruck noch weiter vereinfachen.

$$\left(\left[e^{-\frac{1}{N}}\right]*\left[e^{-\frac{2}{N}}\right]*\ldots*\left[e^{-\frac{r-1}{N}}\right]\right)=e^{-\frac{1}{N}-\frac{2}{N}-\cdots-\frac{r-1}{N}}=e^{-\left(\frac{1}{N}+\frac{2}{N}+\cdots+\frac{r-1}{N}\right)}=e^{-\frac{r(r-1)}{2N}}$$

Wir haben eine untere und eine obere Grenze für P bestimmt und können schreiben:

$$1-e^{-\frac{r(r-1)}{2N}}\leq P\leq\frac{r^2}{2N}-\frac{r}{2N}$$

Da es bei unserer Aufgabenstellung darum geht, ob eine Kollision wahrscheinlich ist, wollen wir wissen, wann P ungefähr 50% ist. Wir berechnen also $\frac{r^2}{2N}-\frac{r}{2N}=\frac{1}{2}$. Das Resultat ist $r=\sqrt{N+\frac{1}{4}}+\frac{1}{2}$ (das negative Resultat interessiert uns nicht, da es keine negative Wahrscheinlichkeit geben kann). Wir rechnen in unserem Beispiel mit sehr hohen Zahlen, das N im SHA-256 Beispiel ist 2^{256}. Wenn die Zahl N so gross ist, spielen in der Formel die Zahlen $\frac{1}{2}$ und $\frac{1}{4}$ keine Rolle. Je grösser also das N ist, desto näher geht r gegen $\sqrt{N}$. Jetzt sehen wir, dass, wenn $r<\sqrt{N}$, P sicherlich kleiner als $\frac{1}{2}$ ist.

Dasselbe machen wir für die untere Grenze. Wir lösen demnach die Gleichung $\frac{1}{2}=1-e^{-\frac{r(r-1)}{2N}}$ nach r auf und erhalten $r=\frac{\sqrt{8*N*\ln(2)+1}+1}{2}$. Dieser Term ist ein wenig weiter von $\sqrt{N}$ entfernt, jedoch nicht sehr viel. Wir wissen aber sicherlich, dass $\sqrt{N}<\frac{\sqrt{8*N*\ln(2)+1}+1}{2}$. Somit wissen wir auch, dass wenn $\sqrt{N}<r$ ist, ist die Wahrscheinlichkeit einer Kollision über 50% liegt. Verallgemeinert kann man somit sagen: Wenn $r<\sqrt{N}$, dann liegt die Wahrscheinlichkeit einer Kollision unter 50%.

Beim SHA-256 Beispiel müssten somit mindestens $r=\sqrt{2^{256}}=2^{128}\sim 3{,}4*10^{38}$ verschiedene Eingabewerte ausprobiert werden, damit eine Kollision wahrscheinlich wäre. Gleich wie

schon zuvor ist die Zahl weitaus grösser, als ein Rechner (2GHz) CPU-Zyklen pro Jahr rechnen kann. Es ginge somit $\frac{3.4*10^{38}}{6.4*10^{16}} = 5.3 * 10^{21}$ Jahre, um eine Kollision festzustellen. [22] [23] [24]

2.2. Der Digital Signature Algorithm

2.2.1. Funktionsweise des Digital Signature Algorithm

Kehren wir wieder zu unserem Eingangsbeispiel, der Bank, zurück. Die Bank muss Transaktionen von einem Konto zu einem anderen festhalten. Damit jedoch jemand von einem Konto aus eine Transaktion tätigen kann, ist es essentiell, dass es sich dabei um den Kontoinhaber des jeweiligen Kontos handelt. Schliesslich könnte sonst jeder zu unserer Bank gehen, behaupten, dass er Eigentümer unseres Kontos sei und sich dann alles Geld vom Konto nehmen. Genau so muss dies auch in der Blockchain umgesetzt sein: es soll unmöglich sein, dass irgendjemand fälschlicherweise behauptet, er sei Eigentümer eines Accounts und dann frei Transaktionen mit den Gütern dieses Accounts betätigen kann.

Möchte Alice tatsächlich in der Bank Geld von ihrem Konto abheben, so wird sie sicherlich zuerst dazu aufgefordert, sich mit einem amtlichen Ausweis auszuweisen (mit einem Pass oder einer ID). Der Bankier vergleicht dann das Foto auf der ID mit dem Gesicht von Alice, welche behauptet, Eigentümer ihres Kontos zu sein. Vermutlich muss sie zusätzlich eine Unterschrift leisten. Nach diesem Schritt wird auf dem Konto nachgesehen, ob der Namen des Eigentümers des Kontos mit dem Namen auf Alices ID übereinstimmt. Ist dies der Fall, so ist Alice dazu berechtigt, Geld von ihrem Konto abzuheben.

Dieser Prozess lässt sich in drei Schritte unterteilen, zwischen denen deutlich unterschieden werden muss.

1. **Identifikation**: Als Alice in die Bank kam und Geld von ihrem Konto abheben wollte, behauptete sie, dass sie Alice sei. Dieser Schritt heisst Identifikation. Es ist wichtig zu sehen, dass sich theoretisch jede Person als Alice identifizieren könnte. Schliesslich kann jeder behaupten, er sei Alice.
2. **Authentifikation**: Nun muss Alice beweisen, dass sie tatsächlich diejenige Person ist, für die sie sich ausgibt. Die ID ist ein Beweis dafür, dass diejenige Person auf dem Bild den Namen trägt, der auf der ID festgehalten ist. Durch Vergleich von Passfoto und Gesicht und der Unterschriften wird sichergestellt, dass Alice tatsächlich die Person ist, für die sie sich ausgibt.
3. **Autorisation**: Zuletzt muss sichergestellt werden, dass Alice berechtigt ist, Geld vom Konto abzuheben. Dies ist genau dann der Fall, wenn Alice als Eigentümer des Kontos angegeben ist und ihr Konto einen positiven Kontostand aufweist.

Zusammengefasst bedeutet **Identifikation** also sich als jemand auszugeben und **Authentifikation** ist die Erbringung des Beweises dafür, dass man diejenige Person ist, für die man sich ausgibt. Durch die **Autorisation** wird entschieden, ob man dazu berechtigt ist, eine Handlung durchzuführen.[25]

Genau diese drei Schritte sollen auch in der Blockchain umgesetzt werden.

Die **Identifikation** ist hierbei ziemlich selbstverständlich: dadurch, dass man von einem Account aus Güter überweisen will, gibt man an, Eigentümer des Accounts zu sein. Die **Authentifikation** ist ein schwieriger Schritt: Bei jeder Transaktion muss für die Knoten aus dem

[22] Das Geburtstagsphänomen, in: staff.uni-mainz. 23.10.2018.
[23] Pommerening/Sergl: Hash-Funktion, 23.10.2018.
[24] Pommerening: Die Sicherheit kryptographischer Verfahren, 23.10.2018.
[25] Miessler: Identification, Authentification and Authorisation, heruntergeladen am 16.03.18

Peer-to-Peer-Netzwerk ersichtlich sein, dass diejenige Person, die behauptet, der Eigentümer des Absender-Accounts zu sein, tatsächlich Eigentümer des Absender-Accounts ist. Ist dieser Schritt gemeistert, so liegt auch die **Autorisation** auf der Hand: Der Eigentümer eines Accounts ist nämlich dazu berechtigt von diesem Account aus Transaktionen zu tätigen.

Es gibt also nur einen Schritt, welcher eine wahre Herausforderung darstellt: Bei jeder Transaktion zu beweisen, dass derjenige, der behauptet, der Eigentümer des Absender-Accounts zu sein, tatsächlich der Eigentümer dieses Accounts ist.

Auf der Blockchain wird dies mit Hilfe des **Digital Signature Algorithm** umgesetzt. Jede Transaktion wird mit diesem Algorithmus signiert. Dadurch ist nachträglich beweisbar, dass die Transaktion signiert wurde. Die Signatur wurde jeweils vom Accountinhaber geleistet. Ausserdem ist diese Signatur je nach Transaktion unterschiedlich und kann somit nicht wiederverwendet werden. Weiterhin darf die Transaktion nach dem Unterschreiben nicht mehr geändert werden, die Signatur wird nämlich nach unerlaubter Änderung an der Transaktion ungültig. Wir werden nun erklären, wie der Digital Signature Algorithm funktioniert.

Jeder Account auf einer Blockchain besitzt einen **privaten Schlüssel**, also eine Art Passwort, welches bei jeder Transaktion von einem Account aus benötigt wird. Dieses Passwort ist streng geheim und sollte niemandem verraten werden oder verloren gehen.

Ein Social Media Account wird beispielsweise durch unseren Namen bezeichnet. Auf der Blockchain ist unsere Accountbezeichnung eine Folge aus Buchstaben und Zahlen. Diese Folge könnte beispielsweise folgendermassen aussehen:

0xbf8393a163f71CedA189ccC1482cC631C4e6eCAF

Dies ist ein **öffentlicher Schlüssel**; er ist für jeden sichtbar. Der öffentliche Schlüssel wird mit Hilfe des privaten Schlüssels generiert, aus dem öffentlichen Schlüssel lässt sich aber nicht auf den privaten Schlüssel schliessen.

Es ist möglich, mit unserem privaten Schlüssel Nachrichten auf eine bestimmte Art und Weise zu verschlüsseln. Auf den genauen Mechanismus werden wir später in diesem Kapitel zurückkommen. Mit dem öffentlichen Schlüssel können wir diese verschlüsselten Nachrichten dann wieder entschlüsseln. Das Entschlüsseln mit dem öffentlichen Schlüssel funktioniert natürlich nur dann, wenn die Originalnachricht durch den zum öffentlichen Schlüssel passenden privaten Schlüssel verschlüsselt wurde.

Das Prinzip des Digital Signature Algorithm sieht folgendermassen aus:

Wollen wir mit unserem Account eine Transaktion tätigen, so halten wir diese Transaktion in einem Dokument fest. Als nächstes ordnen wir diesem Dokument mit dem **SHA-1-Algorithmus** ihren Hash-Wert zu. Nun können wir den Hash-Wert dieses Dokumentes mit unserem Passwort, also unserem privaten Schlüssel, verschlüsseln. Nur wir sind in der Lage diese Verschlüsselung vornehmen, da wir als einzige im Besitz des privaten Schlüssels sind.

Wir veröffentlichen nun sowohl das Dokument als auch den verschlüsselten Hash-Wert des Dokuments. Der öffentliche Schlüssel steckt in dem Namen des Accounts und ist somit ebenfalls für alle ersichtlich. Will nun ein Knoten des Peer-to-Peer Netzwerks überprüfen, ob eine Transaktion tatsächlich vom Inhaber des Accounts getätigt wurde, so kann er den Hash-Wert des Dokuments feststellen und mit dem öffentlichen Schlüssel die Signatur entschlüsseln. Stimmt der Hash-Wert des Dokumentes mit der entschlüsselten Signatur überein, so kann er sichergehen, dass die Transaktion vom Inhaber des Accounts getätigt wurde. Ist dies nicht der Fall, so handelt es sich um einen Betrug.

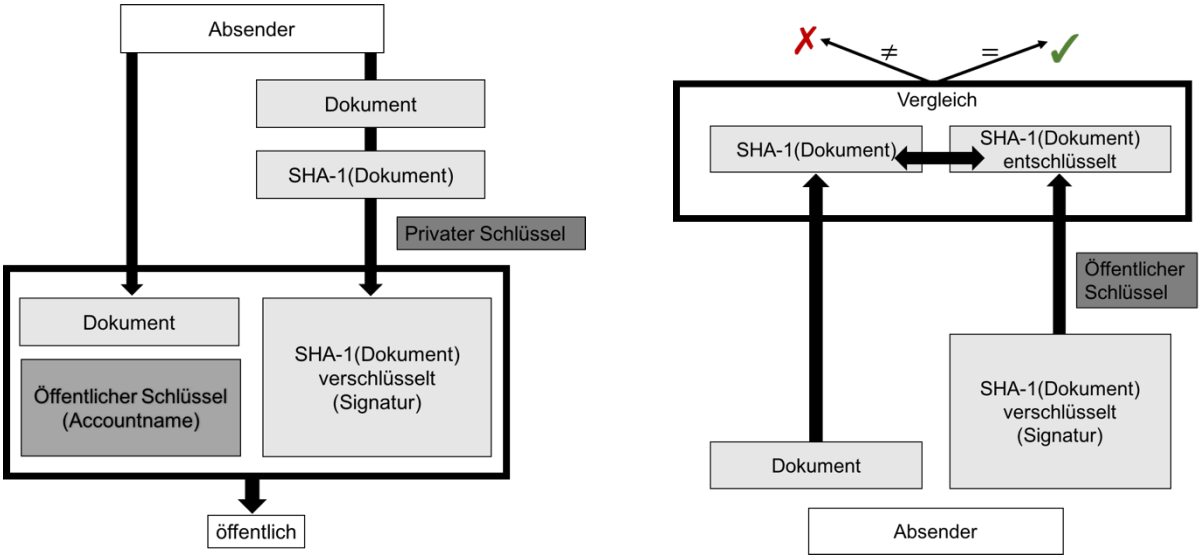

Abbildung 10: Digital Signature Algorithmus

Mit diesem Prinzip wird sofort klar, wieso der private Schlüssel so wichtig ist und er niemals weitergegeben oder verloren werden sollte. Er ist nämlich die einzige Methode zu beweisen, dass wir Inhaber unseres Accounts sind und einverstanden damit sind, von diesem Account aus Transaktionen zu tätigen. Kennt jemand anderes den privaten Schlüssel unseres Accounts, kann er Transaktionen im Namen unseres Accounts tätigen. Geht unser privater Schlüssel verloren, so können wir nicht mehr auf das Eigentum unseres Accounts (unser Geld) zugreifen. Da es in der Blockchain keinen zentralen Knoten gibt, wird es nicht möglich sein, von jemand anderem über unseren Account getätigte Transaktionen rückgängig machen zu lassen oder einen „Passwort vergessen" Button zu drücken, um einen verlorenen privaten Schlüssel erneut zugeschickt zu bekommen. Es ist also essentiell, immer im Hinterkopf zu behalten, wie wichtig der private Schlüssel ist.[26] [27] [28]

2.2.2. Mathematische Grundlagen

Nun, da wir das Prinzip hinter dem Digital Signature Algorithm verstanden haben, möchten wir einen tieferen mathematischen Einblick in diesen überaus spannenden Algorithmus gewinnen. Hierfür sind einige Basiskenntnisse aus mehreren Gebieten der Mathematik nötig. Wir werden diese an dieser Stelle kurz einführen beziehungsweise auffrischen.

Modulorechnen

Beim Modulorechnen wird eine Zahl durch eine andere geteilt. Der Rest, der hierbei entsteht, ist das Resultat.

- $5 : 3 = 1\ Rest\ 2 \quad 5\ mod\ 3 = 2$
- $70 : 9 = 7\ Rest\ 7 \quad 70\ mod\ 9 = 7$

Gruppentheorie

Eine Gruppe ist eine Menge von Elementen $G = \{a_1, a_2, a_3, \dots\}$, welche durch eine Operation $*$ verknüpft sind. Zudem existiert ein **neutrales Element e**, für das gilt $a_i \cdot e = a_i$.

[26] Drescher: Blockchain(Jahr), Kapitel 12f
[27] Digitale Signatur, in: tu-chemnitz, heruntergeladen am 28.03.18
[28] CuriousInventor: Bitcoin, heruntergeladen am 28.03.18

Wird also ein beliebiges Element a_i aus der Gruppe mit dem neutralen Element verknüpft, erhält man wieder das ursprüngliche Element a_i. Weiterhin besitzt jedes Element a_i aus der Gruppe ein **inverses Element**. Wird ein Element mit seiner Inversen verknüpft, erhält man das neutrale Element: $a_i^{-1} \cdot a_i = e$. Innerhalbe einer Gruppe gilt das **Assoziativgesetz** $a_i \cdot (a_j \cdot a_k) = (a_i \cdot a_j) \cdot a_k$.

Auf den ersten Blick wirkt dies vielleicht etwas abstrakt, jedoch operieren wir im Alltag ständig mit Gruppen. Ein Beispiel hierfür sind die ganzen Zahlen bezüglich der Operation +: Die Gruppe $(\mathbb{Z}, +)$. Das neutrale Element ist hierbei 0, das inverse Element jeweils die ganze Zahl mit -1 multipliziert. Uns ist bereits bekannt, dass das Assoziativgesetz bezüglich der Operation + in den ganzen Zahlen gilt.

Eine weitere Gruppe ist $\mathbb{Z}_p \coloneqq \{0,1,2,3, \dots, p-1\}$ also alle Zahlen von 0 bis p-1. Ein prominentes Beispiel sind hier zum Beispiel die ganzen Zahlen, mit denen wir die Uhrzeit zu vollen Stunden angeben: $\mathbb{Z}_{24}$, alle Zahlen von null bis 23 nämlich. Operation in dieser Gruppe wäre die Modulo Addition: $(x + y) \coloneqq (x + y)\ mod\ p$ in $\mathbb{Z}_p$.

Ein Beispiel zur Modulo Addition $\mathbb{Z}_{24}$ ist: $11 + 18 = 5$ (nämlich 29 mod 24).

Für eine Primzahl p versteht man unter $\mathbb{Z}_p^x$ die Menge alle ganzen Zahlen von 1 bis p-1, das ^{x} weist darauf hin, dass die 0 nicht in der Gruppe enthalten ist. Dadurch ist die Modulo-Multiplikation in der Gruppe möglich. In der Modulo-Multiplikation ist 1 das neutrale Element, Jetzt verstehen wir auch, warum Null nicht in der Gruppe entahlten sein kann: Die Null hätte in der Modulo-Multiplikation nämlich keine Inverse; es existiert keine Zahl, welche mit 0 multipliziert 1 ergibt!

Die Modulo-Multiplikation in $\mathbb{Z}_p^x$ funktioniert ähnlich wie die Modulo-Addition, nämlich: $(x \cdot y) \coloneqq (x \cdot y)\ mod\ p$

Als Beispiel betrachten wir hier $\mathbb{Z}_7^x$: Dort ergibt $3 \cdot 6 = 4$.

Ein weiterer wichtiger Begriff ist die **Ordnung** eines Elementes. Die Ordnung eines Elementes g aus der Gruppe G ist das kleinste n, sodass $g^n = e$. Nehmen wir uns wieder die Gruppe $\mathbb{Z}_7^x$ vor: Hier ist 3 die Ordnung von 4. 4 hoch 3 ergibt nämlich (in der Modulo-Multiplikation) 1. Ausserdem sind $4^1, 4^2 \neq 1$.[29] [30]

Der kleine Fermat

Dieser Satz besagt, dass für eine Primzahl p und für ein $a \in \mathbb{Z}_p^x$ gilt, dass $a^{p-1} = 1$ beträgt. Betrachten wir beispielsweise $\mathbb{Z}_7^*$ so gilt

- $1^6 = 1$
- $2^6 = 1$
- $3^6 = 1$
- $4^6 = 1$
- $5^6 = 1$
- $6^6 = 1$

Für diesen Satz gibt es mehrere Beweise, die aber den Rahmen der Arbeit übersteigen würden. Der geneigte Leser findet diese aber in der Literatur.

[29] Rehn: Restklassenringe, heruntergeladen am 07.03.2018
[30] Prof. Dr. Gubler: Lineare Algebra, heruntergeladen am 18.03.2018

Modulo Eigenschaften

Die Homomorphieregel besagt, dass $(a \cdot b) mod\ n = (a\ mod\ n \cdot b\ mod\ n) mod\ n$. Stellt man sich nun vor, dass $x^y mod\ n$ dasselbe ist, wie wenn die Variable x y-mal mit sich selbst multipliziert würde, so kann man einsehen, dass:

$$x^y mod\ n$$
$$= (x \cdot x \cdot x \cdot x \cdot x \cdot \ldots \cdot x) mod\ n$$
$$= (x\ mod\ n \cdot x\ mod\ n \cdot x\ mod\ n \cdot \ldots \cdot x\ mod\ n) mod\ n$$
$$= (x\ mod\ n)^y\ mod\ n$$

So lässt sich aus der Homomorphieregel eine weitere Regel für Potenzen ableiten.[31]

2.2.3. Wie funktioniert der Digital Signature Algorithm genau?

Zahlengeneration

Zuerst wählen wir zwei Primzahlen p und q. Hierbei soll gelten:

$$2^{159} < q < 2^{160}$$
$$2^{511+64j} < p < 2^{512+64j}\ für\ ein\ j \in \{0,1,2,\ldots,8\}$$
$$q|p-1\ (q\ ist\ ein\ Teiler\ von\ p-1)$$

Durch diese Bedingungen hat p mindestens eine Bit Länge von 512, nämlich genau dann, wenn wir für j=0 einsetzen. Die maximale Bit Länge beläuft sich auf 1024, nämlich dann, wenn wir j=8 wählen. Die Bit Länge muss ein Vielfaches von 64 sein, da wir zu der mindesten Bit Länge von 512 jeweils ein Vielfaches von 64 addieren und 512 ein Vielfaches von 64 ist.

Nun suchen wir eine Zahl $g \in \mathbb{Z}p^x$ mit Ordnung q.

- Um dieses g zu finden, suchen wir zuerst eine Zahl $x \in \mathbb{Z}p^x$ mit den Eigenschaften $1 < x < p-1$ und $\left(x^{\frac{p-1}{q}} mod\ p\right) \neq 1$.
- Nun können wir g folgendermassen definieren: $g := x^{\frac{p-1}{q}} mod\ p$. Wir behaupten an dieser Stelle, dass die Ordnung(g) = q, die vorhin verlangte Eigenschaft also erfüllt ist.
- Laut oben beschriebener Definition ist $g := x^{\frac{p-1}{q}} mod\ p$.
- Potenzieren wir nun auf beiden Seiten mit q erhalten wir die Gleichung $g^q = (x^{\frac{p-1}{q}} mod\ p)^q$.
- Mit der im Einstieg gezeigten Umformmethode zu Exponenten bei Modulo Rechnungen formen wir diese Gleichung weiterhin um: $g^q = (x^{\frac{p-1}{q}})^q\ mod\ p = x^{p-1}\ mod\ p$.
- Der „kleine Fermat“ besagt gerade, dass $x^{p-1}\ mod\ p = 1$, somit ist $g^q = 1$.
- Nun können wir sicher sein, dass die Ordnung(g) höchstens q ist. An dieser Stelle müssen wir deswegen nur noch sicherstellen, dass die Ordnung(g) = q ist.
- Wäre die Ordnung von g gleich r für ein r<q, dann liesse sich q als $r \cdot k + x$ darstellen für ein k und x aus den natürlichen Zahl. Es gilt weiterhin $g^r mod\ n = 1$ und $g^r mod\ n = (g^{r \cdot k+x}) mod\ n = \left((g^{r \cdot k}) mod\ n \cdot (g^x) mod\ n\right) mod\ n = 1$.

[31] Ganter: Modulo, heruntergeladen am 08.10.18

$(g^{r \cdot k}) mod\ n$ beträgt hierbei sicherlich 1. Weiterhin muss $(g^x) mod\ n = 1$ betragen. Dies kann nur gelten für ein x = 0. Schliesslich wird angenommen, dass r die Ordnung von g ist und somit für kein x<r und ungleich Null $(g^x) mod\ n = 1$ gelten darf.

- r ist aus oben ausgeführtem Grund ein Teiler von q sein**.** Da q jedoch eine Primzahl ist, besitzt q keine Teiler. Somit stimmt unsere Behauptung von vorhin.

Das Tripel (p, q, g) wird nun veröffentlicht. Will jemand ein Dokument signieren, so wählt er eine private Zahl $a \in \mathbb{Z}q$ und berechnet $A \coloneqq g^a\ mod\ p$. A wird ebenfalls veröffentlicht, a bleibt jedoch privat. Dem öffentlichen Schlüssel entsprechen also p, q, g und A. a ist unser privater Schlüssel.

Generieren der Signatur

Wir wollen nun das Dokument x signieren. Die Bit Länge von x darf beliebig sein. Durch den Hashalgorithmus SHA-1 wird diesem beliebig grossen Dokument eine Bit Länge von 160 zugeordnet. Der SHA-1 funktioniert nach demselben Prinzip wie der SHA-256, den wir im vorherigen Kapitel bereits kennengelernt haben. Wir bezeichnen den SHA-1 eines Dokuments z hier als Funktion h(z).

Wir suchen ein zufälliges $k \in \{1, 2, \dots, q-1\}$. Nun bestimmen wir $r \coloneqq (g^k mod\ p)\ mod\ q$ und $s \coloneqq k^{-1}(h(x) + a \cdot r)\ mod\ q$.

Das Tupel (r, s) ist die Signatur von x. s entspricht hierbei dem verschlüsselten Hashwert des Dokuments. Würde das Dokument verändert (Transaktionen verändert), änderte sich s.

Verifikation

Bei der Verifikation wird ein Ausdruck, den wir hier (*) nennen, berechnet.

$$(*) \coloneqq \left(\left(g^{(s^{-1} \cdot h(x) mod\ q} \cdot A^{(r \cdot s^{-1}) mod\ q}\right) mod\ p\right) mod\ q$$

Es wird überprüft, ob der Ausdruck (*) r beträgt oder nicht. Trifft dies zu, so ist das Dokument vom Inhaber des privaten Schlüssels signiert worden, falls nicht, dann ist dies nicht der Fall.

Doch wieso wird die Verifikation auf diese Art durchgeführt?

Laut Definition von A ist $A \coloneqq g^a\ mod\ p$

$A^{(r \cdot s^{-1}) mod\ q} = (g^a\ mod\ p)^{(r \cdot s^{-1}) mod\ q}$, wenn für A der obige Ausdruck eingesetzt wird

$(g^a\ mod\ p)^{(r \cdot s^{-1}) mod\ q} = (g^a)^{(r \cdot s^{-1}) mod\ q}\ mod\ p$ durch die Modulogesetze

$= g^{(a \cdot r \cdot s^{-1}) mod\ q}\ mod\ p$ durch Anwenden der Potenzgesetze

Dies können wir nun in (*) für $A^{(r \cdot s^{-1}) mod\ q}$ einsetzen.

$$(*) \coloneqq \left(\left(g^{(s^{-1} \cdot h(x) mod\ q} \cdot A^{(r \cdot s^{-1}) mod\ q}\right) mod\ p\right) mod\ q$$

$$= \left(\left(g^{(s^{-1} \cdot h(x) mod\ q} \cdot g^{(a \cdot r \cdot s^{-1}) mod\ q}\right) mod\ p\right) mod\ q$$

$= \left(\left(g^{s^{-1} \cdot (h(x) + a \cdot r) mod\ q}\right) mod\ p\right) mod\ q$ durch Anwenden von Potenzgesetzen und Ausklammern

Wir hatten unser s definiert als $s \coloneqq k^{-1}(h(x) + a \cdot r)\ mod\ q$. Durch eine Multiplikation mit k ergibt sich folgender Ausdruck:

$$s \cdot k = (h(x) + a \cdot r)\ mod\ q$$

Deswegen ersetzen wir in (*) den Ausdruck $(h(x) + a \cdot r)\ mod\ q$ durch $s \cdot k$.

$$(*) = \left(\left(g^{s^{-1}\cdot(h(x)+a\cdot r)mod\ q}\right)mod\ p\right) mod\ q$$

$$= \left(\left(g^{s^{-1}\cdot s\cdot k}\right)mod\ p\right) mod\ q$$

Es gilt:

$$s \cdot s^{-1} = 1$$

Eingesetzt in (*) entsteht

$$(*) = \left(\left(g^{s^{-1}\cdot s\cdot k}\right)mod\ p\right) mod\ q$$

$$= \left((g^{k})mod\ p\right)mod\ q$$

$$= r \text{ laut Definition von r}$$

Diese Umformung funktioniert natürlich nur, wenn die Verschlüsselung, wie oben definiert, mit dem passenden privaten Schlüssel vorgenommen wurde.

Die obigen Umformungen entsprechen dem Vergleichen von Signatur und Hashwert des Dokuments. Mit Hilfe dieser Umformungen wird nämlich gezeigt, dass die Signatur (s und r) tatsächlich durch den zum öffentlichen Schlüssel gehörenden privaten Schlüssel und dem Hashwert der angegebenen Transaktionen erstellt wurde.[32]

2.3. Fragen

4. In dieser Aufgabe geht es darum, die ersten Schritte des SHA-256 von Hand durchzuführen. Als Eingabewert dient das Wort «Blockchain».
   a) Wie sieht «Blockchain» als Binärzahl codiert aus? Hierbei soll die ASCII-Tabelle für die Codierung verwendet werden (siehe Abbildung 8).
   b) Als nächstes werden 16 „Wörter“ mit 32 Bit Länge gebildet. Nicht zu vergessen ist hierbei, dass am Schluss die Länge des Wortes in Bit anzuhängen ist.
   c) Wie sieht das 17. Wort aus?
   d) Wie sieht H1 aus?
5. Wir stellen uns eine Funktion vor, die einem Eingabewert zufällig eine 6-stellige Hexadezimalzahl zuordnet. Wie viele Eingabewerte müssten mindestens ausprobiert werden, sodass die Wahrscheinlichkeit bei 50% liegt, dass eine Kollision entsteht?
6. Weshalb ist es extrem wichtig den privaten Schlüssel geheim zu halten und niemals zu verlieren?
7. Wir betrachten die Gesamtheit der reellen Zahlen bezüglich der Multiplikation.
   a) Handelt es sich um eine Gruppe?
   b) Was wäre eine (weitere?) Gruppe bezüglich der Multiplikation?
8. Wir betrachten die Gruppe $\mathbb{Z}_{13}^{x}$. Was ist nun die Ordnung von 9?

[32] Buchmann: Digital Signature Algorithm, heruntergeladen am 16.03.2018

3. FUNKTIONSWEISE DER BLOCKCHAIN

3.1. Festhalten von Transaktionen

Da wir nun sichergestellt haben, dass nur der tatsächliche Kontoinhaber eines Kontos Aktionen von diesem Konto aus tätigen kann, können wir einen Schritt weitergehen.

Führt jemand eine Transaktion von seinem Konto auf ein anderes durch, so muss dies von der Bank festgehalten werden. Die Bank setzt dies um, indem sie den zu überweisenden Betrag beim Kontostand des ersten Kontos abzieht und dann zum Kontostand des zweiten Kontos addiert.

Würde die Blockchain diesen Schritt auf äquivalente Art umsetzen, so müssten ständig alle Kontostände abgespeichert werden. Auf der Blockchain werden deswegen lediglich die Änderungen verzeichnet, also direkt die Transaktionen. Eine Transaktion umfasst drei Informationen, die festgehalten werden müssen: Absender, Empfänger und überwiesener Betrag.[33]

Möchte Alice beispielsweise 5 Bitcoins an Bob überweisen, so wird auf der Blockchain Alice als Absender, Bob als Empfänger und 5 Bitcoins als überwiesener Geldbetrag festgehalten.

Betrag [BTC]	5	Von	Alice	→	Bob

Bob möchte nun ebenfalls eine Transaktion tätigen und zwar möchte er 2 Bitcoins an Edward überweisen. Er muss nun nachweisen, dass eine Transaktion an ihn existiert, welche 2 Bitcoins umfasst. Diese Transaktion darf natürlich noch nicht wieder ausgegeben worden sein. Bob wählt hierfür die Transaktion von Alice.

In dieser hat er jedoch 5 Bitcoins und nicht nur 2 erhalten. Bob wählt deswegen diese Transaktion, überweist 2 Bitcoins davon an Edward und die restlichen 3 erneut an sich selbst. Diese 3 Bitcoins kann Bob nun in einer weiteren Transaktion zu einem späteren Zeitpunkt ausgeben.

Betrag [BTC] Betrag [BTC]	2 3	Von Von	Bob Bob	→ →	Edward Bob

Edward hat zusätzlich zu Bobs Transaktion früher bereits eine Transaktion von 2 Bitcoins von Daniel erhalten.

Betrag [BTC] Betrag [BTC]	2 1.8	Von Von	Daniel Daniel	→ →	Edward Daniel

[33] CuriousInventor: Bitcoin, heruntergeladen am 28.03.18

Er möchte nun eine Transaktion von 3.5 Bitcoins an Chantal tätigen. Da keine Transaktion an Edward über mindestens 3.5 Bitcoins vorliegt, verwendet er mehrere Transaktionen. Zuerst gibt er seine gesamten 2 Bitcoins, welche er von Bob erhalten hat, an Chantal aus. Nun verwendet er 1.5 Bitcoins der 2 Bitcoins von Daniels Transaktion. Die restlichen 0.5 Bitcoins überweist er erneut an sich selbst.

Betrag [BTC]	2	Von	Edward	→	Chantal
Betrag [BTC]	1.5	Von	Edward	→	Chantal
Betrag [BTC]	0.5	Von	Edward	→	Edward

Auf diese Art und Weise können Transaktionen auf der Blockchain festgehalten werden, wobei stets nur Änderungen an Kontoständen aufgezeichnet werden.

Möchte man seinen Kontostand bestimmen, so muss man lediglich alle noch nicht ausgegebenen Transaktionsbeträge an sich zusammenzählen.

Die Transaktionen werden wie im Eingangsbeispiel der Yapis im Netzwerk verteilt. Die Liste an Transaktionen, welche verteilt wird, heisst „**Distributed Ledger**“. Ledger bedeutet Grundbuch, Dies ist dadurch motiviert, dassein Notar im Grundbuch die beglaubigten Eigentumsübertragungen verzeichnet. Distributed Ledger heisst es deshalb, weil es nicht ein zentrales Grundbuch, sondern viele Grundbücher verteilt über das Netzwerk gibt. Im Anschluss werden wir sehen, wie der Distributed Ledger im Netzwerk verteilt werden kann und wie Betrug hierbei verunmöglicht wird.

3.2. Aufbau der Blockchain

3.2.1. Aufbau der Blöcke

Die Blockchain besteht, wie der Name bereits sagt, aus vielen Blöcken. Die Blöcke sind nummeriert (beginnend bei Block eins). Es liegt also eine bestimmte Reihenfolge von Blöcken vor. In jedem Block sind als Inhalt eine Liste von Transaktionen, wie sie im vorherigen Kapitel beschrieben sind, festgehalten. Ausserdem enthält jeder Block einen Zeitstempel, also die exakte Uhrzeit und das Datum des Zeitpunkts, zu dem er hinzugefügt worden ist. Die Menge aller Blöcke wird als **Distributed Ledger** bezeichnet.

Für eine Bank genügt es, Transaktionen festzuhalten. Schliesslich kann nur sie auf Transaktionen zugreifen und diese verändern. Es ist also praktisch unmöglich, dass unehrliche Personen willkürlich Transaktionen verändern.

Die Transaktionen der Blockchain jedoch werden in einem öffentlichen Peer-to-Peer Netzwerk gespeichert. Theoretisch könnte also jeder eine beliebige Transaktion verändern. Es muss dementsprechend ein Konzept entwickelt werden, damit dieser Fall nicht eintritt.

Dafür wird der Hashwert jedes Blocks berechnet. Dieser wird dann im jeweiligen Block festgehalten. Den Hashwert erhält man, indem man aus allen Inhalten (also der Liste der Transaktionen, dem Zeitstempel und der Blocknummer) des Blocks den SHA-256-Wert bildet.

Wird im Inhalt vom Block etwas verändert, so ist der Hashwert des Blockes ein komplett neuer. Dies ist darauf zurückzuführen, dass die Hashfunktionen Pseudezufallsfunktionen sind und somit jede Änderung im Eingabewert zu einem veränderten Ausgabewert führt.

Ein zusätzlicher Wert namens **Referenzwert** ist in jedem Block enthalten. Der Referenzwert ist der Hashwert des vorherigen Blocks. Auch er wird zur Berechnung des jeweiligen Hashwerts miteinbezogen. Der allererste Block besitzt keinen vorherigen Block. Deswegen betragen hier beim Referenzwert alle Ziffern Null.

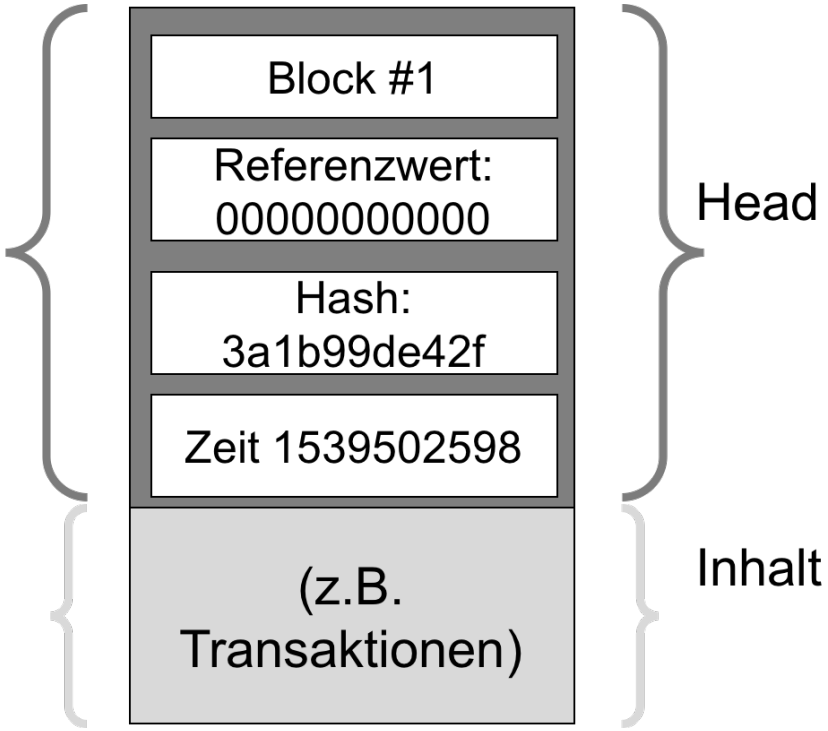

Abbildung 11: Block einer Blockchain

Der Eingabewert für die SHA-256 Funktion besteht nun also zusätzlich zur Liste an Transaktionen, dem Zeitstempel und der Blocknummer aus dem Referenzwert. In der folgenden Grafik sind drei aufeinanderfolgende Blöcke zu sehen.

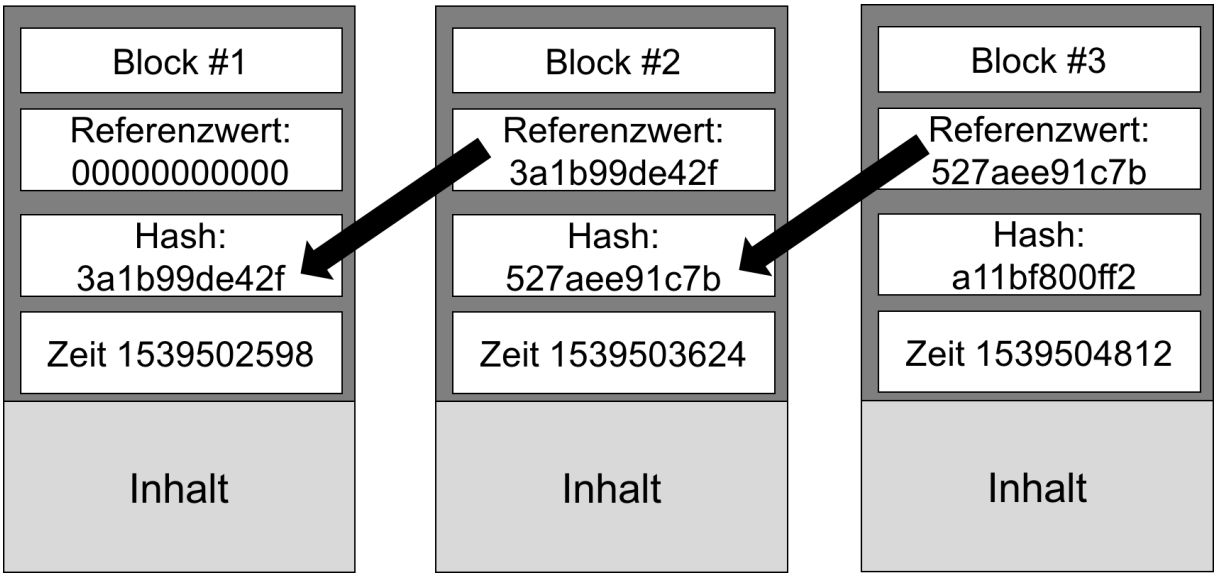

Abbildung 12: Verkettung von Blöcken über den Hashwert

Wird nun eine Veränderung in einem Block vorgenommen, so verändert sich der Hashwert dieses Blocks. Dadurch verändert sich der Referenzwert des nächsten Blocks. Da der Referenzwert in der Bildung des Hashwerts eines Blocks inbegriffen ist, verändert sich auch der Hashwert des nächsten Blocks. Es erfolgt eine Veränderung in allen darauffolgenden Blöcken.

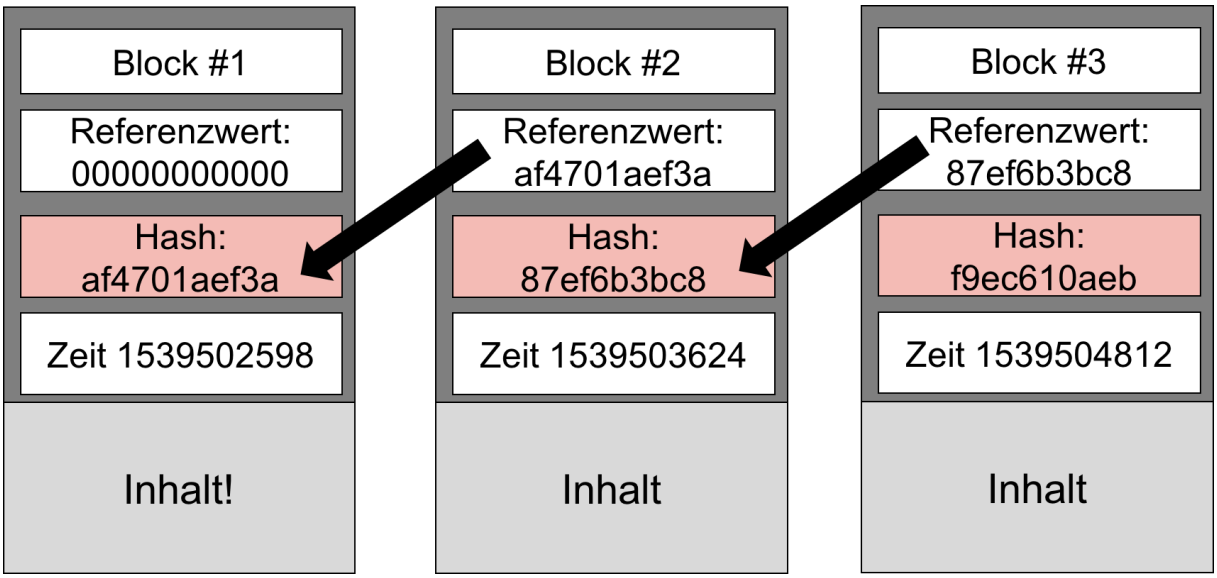

Abbildung 13: Aufbrechen der Verkettung durch Manipulation

In Abbildung 13 wurde der Inhalt des Blocks eins zu Inhalt! geändert. Dadurch verändert sich der Hashwert des Blocks eins, dadurch wiederum der Referenzwert von Block zwei und drei. Da der Referenzwert in der Bildung des Hashwerts eines Blocks inbegriffen ist, verändert sich auch der Hashwert von Block zwei und drei. Die Veränderung findet auf diese Art in allen folgenden Blöcken statt.

Nun wird der Name Blockchain endgültig klar: Die vielen Blöcke sind durch die Referenzwerte wie eine Kette miteinander verbunden.

Momentan führt eine Veränderung des Inhalts eines Blockes zu Veränderungen in allen weiteren Blöcken. Weitere Konsequenzen sind bisher noch nicht vorhanden. Damit das Verändern eines Blockes unmöglich wird, wird ein Rätsel zur Blockchain hinzugefügt. Um einen neuen Block an die Blockchain anzuhängen, muss das Rätsel gelöst werden. Das Rätsel zu lösen benötigt eine gewisse Zeit. Wird eine Veränderung an einem alten Block vorgenommen, so verändern sich die Lösungen aller Rätsel des Blockes und der darauffolgenden Blöcke. Um eine korrekte Blockchain zu erhalten, müssen alle Rätsel also wieder neu gelöst werden, was einen Zeitaufwand zur Folge hat.

Das zu lösende Rätsel lässt sich gut mit einem Fahrradschloss vergleichen. An jedem Block befindet sich ein Fahrradschloss. Dieses muss durch Finden einer richtigen Zahlenkombination geöffnet werden. Wird in einem Block etwas verändert, so verschliessen sich alle Fahrradschlösser dieses Blocks und der darauffolgenden Blöcke und neue Zahlenkombinationen müssen gefunden werden.

Doch wie sieht dieses Rätsel bei der Blockchain konkret aus? In jedem Block wird ein zusätzlicher Wert, der **Nonce**, festgehalten. Der Nonce ist eine Zahl, die ebenfalls in der Berechnung des Hashwertes eines Blocks einbezogen wird.

Der Nonce muss nun solange verändert werden, bis eine bestimmte Bedingung erfüllt ist, also beispielsweise bis die ersten n Stellen des Hashwerts des jeweiligen Blockes Null betragen. Die Anzahl n der Nullen variieren hierbei. Umso mehr Nullen nötig sind, desto geringer ist

die Wahrscheinlichkeit die Bedingung zu erfüllen, desto mehr Ausprobieren ist also nötig, um den passenden Nonce zu finden. Dieser Vorgang nennt sich **„Proof of Work“**. Jedes Mal, wenn ein neuer Block angehängt wird, muss dieser Proof of Work durchgeführt werden, dieser ist mit einem Zeitaufwand verbunden. Dadurch, dass der passende Nonce nur durch Ausprobieren gefunden werden kann, wird durch das Vorhandensein des Nonces bewiesen, dass ein Computer eine zeitaufwändige Arbeit durchgeführt hat, um einen neuen Block anzuhängen.

Der Proof of Work bei der Blockchain von Bitcoin wird so aufwändig gestaltet, dass er etwa zehn Minuten in Anspruch nimmt. Dabei muss die Anzahl der Nullen immer wieder erhöht werden, da die Rechengeschwindigkeit von Computern ständig zunimmt.

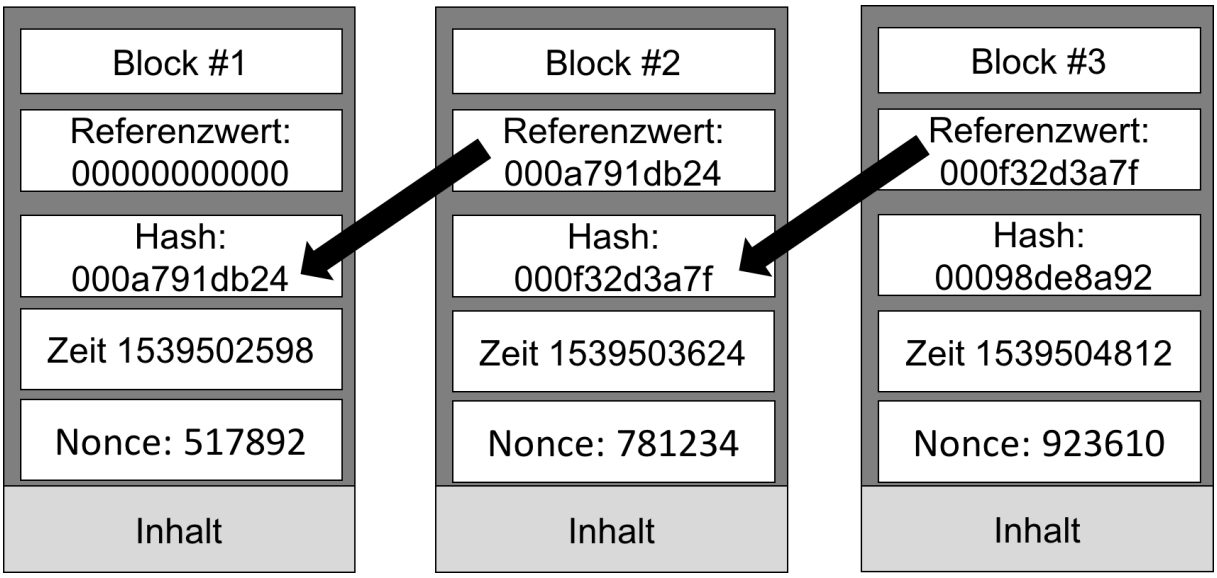

Abbildung 14: Blöcke der Blockchain mit proof-of-work (Nonce)

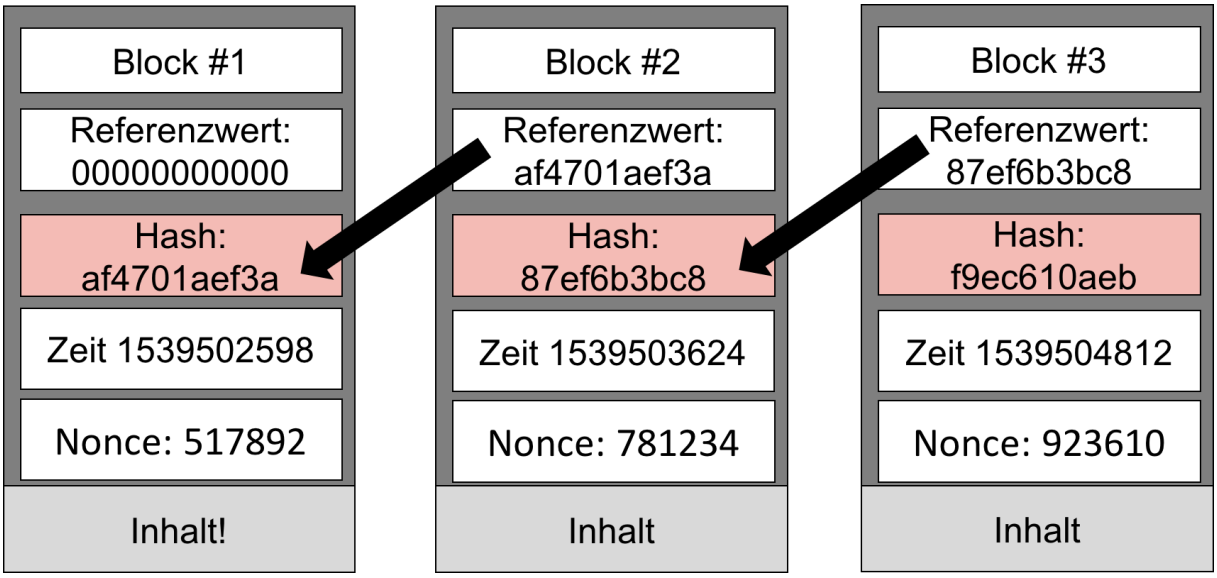

Abbildung 15: Manipulation auf der Blockchain, Verhalten des Nonces.

Der richtige Nonce lässt sich wirklich nur durch Ausprobieren finden. Wieso ist dies so? Der Grund hierfür ist die Unumkehrbarkeit der Hashfunktionen. Dadurch lässt sich nicht von einem Hashwert auf den Ursprungswert schliessen, sondern Ursprungswerte müssen ausprobiert werden, bis der gewünschte Hashwert gefunden ist.

Wir haben vorhin herausgefunden, dass eine Veränderung im Inhalt eines Blocks eine Veränderung der Hashwerte bei diesem und den darauffolgenden Blöcken zur Folge hat. Dies hat unmittelbar zur Folge, dass die Nullen in den ersten paar Stellen verloren gehen. Der Proof of Work muss also sowohl bei diesem als auch bei allen weiteren Blöcken aufs Neue verrichtet werden muss. Dies wiederum ist mit einem riesigen Zeitaufwand verbunden. In dieser Zeitspanne haben andere Computer im Netzwerk bereits eine grosse Anzahl neuer Blöcke angehängt. Der veränderte Ledger ist also viel kürzer als der unveränderte. Dies ist ein klares Unterscheidungsmerkmal. Die veränderte Blockchain wird dadurch sofort erkannt und von der unveränderten unterschieden. Zusammengefasst ist also die längste Blockchain die gültige.

Nun, da wir verstanden haben, wie die Blockchain funktioniert, ist es wichtig zu verstehen, auf welche Art neue Blöcke an angehängt werden.

3.2.2. Anhängen neuer Blöcke

Das Anhängen von neuen Blöcken nennt sich auch „**Mining**“. Alle zu tätigenden Transaktionen werden in einem „Pool“ festgehalten. Die Computer im Peer-To-Peer Netzwerk suchen sich aus diesen Transaktionen einige für den Inhalt ihres Blockes aus. Dabei müssen die Transaktionen durch den jeweiligen Computer verifiziert werden. Dies bedeutet erstens, dass der Computer sicherstellen muss, dass der Sender der Transaktion über die nötigen Ressourcen verfügt. Will jemand 5 Bitcoins versenden so ist es nötig, dass eine an ihn gesendete, nicht ausgegebene Transaktion über 5 Bitcoins existiert.

Nun geht es darum, den Proof of Work für den neuen Block zu absolvieren. Dieser ist sehr rechen- und zeitaufwändig. Ist der passende Nonce gefunden, so wird die neue Version des Ledgers, welche einen weiteren Block enthält, im Netzwerk verbreitet.

Mining ist jeweils damit verbunden, den Proof of Work zu verrichten. Dieser benötigt wegen der vielen Berechnungen von Hashwerten eine Menge an Energie. Kein Knoten im Peer-to-Peer-Netzwerk würde diese Energie freiwillig zur Verfügung stellen. Deswegen werden Belohnungen an diejenigen gegeben, welche ihre Rechenleistung zur Verfügung stellen. Jeder an den Ledger angehängte Block wird dementsprechend durch eine gewisse Summe an Kryptowährung belohnt. Wie funktioniert dies? Die einzelnen Knoten stehen im stetigen Wettkampf miteinander. Sobald ein neuer Block an den Ledger angehängt wurde, wird die neue Version des Ledgers weiterverbreitet. Da dies von allen Knoten umgesetzt wird, sind viele Versionen des Ledgers im Umlauf. Ein Knoten verbreitet immer die Version weiter, welche am längsten ist, dementsprechend am meisten Zeit in Anspruch genommen hat und somit keine Veränderungen an früheren Blöcken enthält. Ausserdem müssen die Signaturen zu allen neu angefügten Transaktionen korrekt sein. Diejenigen, welche bei diesem Ledger einen Block angehängt haben, bekommen als Belohnung eine gewisse Summe an Kryptowährung. Betrügt ein Knoten, so wird seine Version des Ledgers sicherlich nicht weiterverbreitet. Diesem Knoten bleibt deswegen die Belohnung aus, obwohl er viel Energie aufgewendet hat. Betrug hat somit nicht nur keine Chance, sondern ist sogar mit einem Energieaufwand und damit finanziellem Verlust verbunden. [34] [35] [36]

Ein wichtiger Teil der Funktionsweise der Blockchain ist, dass Versionen des Ledgers ständig im Peer-to-Peer-Netzwerk verbreitet werden. Wie dieser Aspekt funktioniert, werden wir uns im nächsten Schritt ansehen.

[34] Simply Explained – Savjee: Blockchain, heruntergeladen am 28.03.18
[35] Brownworth: Blockchain, heruntergeladen am 30.05.18
[36] Drescher: Blockchain (2017), Kapitel 14ff

3.3. Verteilen des Distributed Ledgers im Peer-to-Peer Netzwerk

3.3.1. Gossip-Prinzip

Das Prinzip der Verteilung des Ledgers lässt sich durch einen Vergleich zum menschlichen Phänomen der sozialen Interaktion und Kommunikation gut erklären. Wenn Informationen unter Freunden oder Arbeitskollegen ausgetauscht werden, ist dabei die Wahrscheinlichkeit hoch, dass diese Informationen an weitere Freunde weitergegeben werden. Dabei ist die Wahrscheinlichkeit meist noch höher, wenn die Person darum gebeten wird, die Information nicht weiterzuerzählen. Diese Tatsache lässt sich dadurch begründen, dass der Austausch von Informationen über Andere normalerweise eine relativ effiziente Methode ist, soziale Kontakte zu stärken oder neue Freunde zu finden. Ausserdem sind Menschen an Informationen über die Leute in ihrem Umfeld interessiert. Deshalb findet ein grosser Informationsaustausch statt. Die Informationen werden hierbei jeweils von Einzelpersonen verbreitet.

Die Gespräche, die zur Verbreitung von Informationen geführt werden, lassen sich in drei Kategorien unterteilen. Small Talk sind Gespräche, welche keine wichtigen oder neuen Informationen enthalten. Sie könnten beispielsweise über das Wetter oder die Vorhaben am nächsten Wochenende handeln. Trotzdem sind diese Gespräche sehr wichtig, da sie dazu dienen Beziehungen aufrechtzuerhalten. Eine weitere Kategorie sind Neuigkeiten. Hierbei werden spannende Neuigkeiten mit wichtigem Inhalt ausgetauscht. Ein Beispiel hierfür wäre, dass jemand gerade erst ein neues Haus gekauft hat. Letzte Kategorie sind Gespräche mit noch unbekannten Leuten, um Bekanntschaften zu knüpfen und neue Freunde zu finden. Beispielweise, indem man beim Einkaufen jemanden anspricht um diese Person näher kennenzulernen.

3.3.2. Anwendung im Peer-to-Peer Netzwerk

Im letzten Kapitel haben wir näher betrachtet, dass die Verbreitung von aktuellen Versionen des Distributed Ledgers im Peer-to-Peer-Netzwerk ein zentrales Element der Funktionsweise der Blockchain Technologie ist. Die Verbreitung der aktuellen Versionen des Distributed Ledgers lässt sich sehr gut an Hand des oben erläuterten Beispiels der Informationsverbreitung in einem Freundeskreis erklären. Im Peer-to-Peer Netzwerk gibt es keinen zentralen Computer. Die Informationen können nicht von einem Element aus an alle weiteren Computer verschickt werden. Die Grundidee hinter der Verbreitung von Information im Peer-to-Peer ist dieselbe wie diejenige im Beispiel zur menschlichen Kommunikation. Die Computer im Peer-to-Peer Netzwerk haben jeweils „Freunde“. Als „Freunde“ bezeichnen wir diejenigen Computer, mit welchen der einzelne Computer Informationen austauscht. Zwischen Computern gibt es ebenfalls die drei Gesprächsarten, welche wir bezüglich des Austauschs von Informationen zwischen Menschen kategorisiert haben.

Computer führen Small Talk, indem sie den anderen Computern kurze Nachrichten (Ping genannt) an ihre befreundeten Computer schicken. Die Empfänger antworten auf diese Ping Nachrichten durch einen Pong. Diese dienen dazu die verfügbaren Computer zu identifizieren und die Verbindung zu ihnen aufrecht zu erhalten. Auch lassen sich hierdurch Computer identifizieren, welche offline gegangen sind, da sie nicht mit Pong antworten. Solche Computer können dann als „Freunde“ entfernt werden. Neuigkeiten werden an alle befreundeten Computer weitergegeben. Bei den Neuigkeiten handelt es sich um aktuelle Versionen des Distributed Ledgers. Die Empfänger vergleichen diese aktuellen Versionen mit ihrer und fügen die neuen Blöcke bei ihrer Version hinzu, falls neue Blöcke enthalten sind. Die aktuelle Version wird dann weiterhin an alle Computer im Peer-to-Peer Netzwerk versendet. Wenn ein Computer, welcher zuvor offline war, wieder online ist, so informiert er andere Computer im Netzwerk darüber. Er sendet ihnen eine Anfrage, welche diese dann annehmen können. Das-

selbe gilt für Computer, die neu im Peer-to-Peer Netzwerk sind. Diese Computer werden dann über die aktuelle Version des Distributed Ledgers informiert. Diese Art von Gespräch kann mit dem Schliessen neuer Bekanntschaften verglichen werden.[37]

Im Unterschied zur gesprochenen und geschriebenen Sprache der Menschen kommunizieren Computer über ein digitales Netzwerk. Das dabei meistverwendete Netzwerk ist das Internet. Daher bietet es sich für die Realisierung der Blockchain Technologie an, dass die Computer über das Internet kommunizieren. Voraussetzung dafür, dass über das Internet kommuniziert werden kann, ist, dass alle Computer über das Internet miteinander verbunden sind. Auch besitzt jeder Computer eine einzigartige Adresse, damit er identifiziert werden kann. Mit dieser Adresse ist es ihm dann möglich, Nachrichten im Peer-to-Peer Netzwerk zu versenden und zu empfangen.

Abbildung 16: Gossip Prinzip. Rockwell: The Gossips, 10.10.2018

Es gibt einige Schwierigkeiten, welche gemeistert werden müssen. Nachrichten könnten nämlich verloren gehen, in falscher Reihenfolge oder mehr als nur einmal ankommen. Dies sind Kommunikationsprobleme, welche jedoch elegant gelöst werden können.

Jeder Computer hat eine eigene Liste von Computern („Freunden“), mit welchen er kommuniziert. Dabei hat aber jeder dieser Computer wieder eigene „Freunde“, mit denen er kommuniziert. Eine Information wird somit zuerst an die eigenen „Freunde“ verschickt, diese wiederum versenden die Nachricht weiter an eine Anzahl von Computern, welche mit dem ursprünglichen Computer nichts zu tun haben. Bei Menschen kann dies mit einer Firma verglichen werden. Wenn sich eine Person mit ihren gewohnten Arbeitskollegen austauscht, sind noch nicht alle Mitarbeiter einer Firma informiert. Die engen Arbeitskollegen können jedoch die Information an weitere Mitarbeiter geben, mit welchen die Ursprungsperson nicht in Kontakt ist. Wenn ein Computer eine Nachricht erhält, sendet er diese an alle Peers weiter, welche bei ihm auf der Liste seiner „Freunde“ stehen. Diese Computer machen dann wieder dasselbe mit ihrer Liste von „Freunden“. Die Information wird dabei auf sicherem Weg an jeden individuellen Computer verbreitet. Dieses Prinzip wird Gossip Prinzip genannt, da es mit dem Klatsch einer Menschengruppe vergleichbar ist. Das wahrscheinlich erste Mal, dass dieses Prinzip verwendet wurde, war in Alan Demers „Epidemic Algorithms for Replicated Database Maintenance“. Demers arbeitete allerdings nicht mit dem Begriff Gossip, sondern mit dem Begriff Rumors, also Gerüchte. [38] [39]

[37] Gossip, in: Hyperledger Fabric, heruntergeladen am 22.10.2018
[38] Demers/Gealy/Greene/Hauser/Irish/Larson/Manning/Shenker/ Sturgis/Swinehart/ Terry/ Woods: Epidemic Algorithms (1989), S.8

Jede Nachricht, welche zwischen den Computern versendet wird, hat einen eigenen Hash-Wert. Damit können doppelt erhaltene Nachrichten leicht identifiziert und ignoriert werden. Da jede Nachricht mit einem Zeitstempel versehen wird, ist auch das Problem der zeitlichen Verschiebung elegant gelöst.

Stellen wir uns nun vor, dass in einer Firma ein Gerücht erzählt wird. Alice erfährt das Gerücht durch einen Arbeitskollegen und erzählt es weiter. Nun ist es möglich, dass sie das Gerücht erneut durch einen anderen Mitarbeiter erfährt. Sie wird dieses Gerücht nicht mehr weitererzählen. Dasselbe wird bei Computern umgesetzt. Wenn ein Computer eine Nachricht ein zweites Mal erhält, sendet er diese nicht nochmal weiter, denn alle seine Peers haben diese ja schon erhalten. So wird eine Nachricht nicht für immer im Netzwerk weiterversendet. [40]

Durch all die in diesem Kapitel beschriebenen Prozesse ist es möglich, Informationen effizient im Peer-to-Peer Netzwerk zu verbreiten und sicherzustellen, dass keine wichtigen Informationen verloren gehen. Als letztes werden wir uns Probleme ansehen, die durch die beschriebene Blockchain Technologie entstehen und wie sie gelöst werden können.

3.4. Merkle Trees

Ein weiteres wichtiges Element der Blockchain Technologie ist der Merkle Tree, welchen wir hier genauer kennenlernen werden. Ein Merkle Tree ist eine Datenstruktur. In dieser Datenstruktur ist es einfacher möglich, die Gültigkeit der einzelnen Daten zu bestimmen.

Stellen wir uns vor, es bestehen vier Transaktionen. All diese Transaktionen (in Realität weitaus mehr als vier) sind in einem Block enthalten. Um zu sehen, ob eine spezifische Transaktion in einem Block enthalten ist oder ob eine Transaktion nicht verändert wurde, müsste diese unter allen Transaktionen eines Blockes gesucht werden und dann der Hashwert des gesamten Blocks mit allen Transaktionen gebildet werden. Dies wäre ein sehr grosser Aufwand.

Dieser Prozess kann deutlich vereinfacht werden. Wir stellen uns vor, Alice sendet eine gewisse Menge einer Kryptowährung an Bob. Alice will nun wissen, ob die Transaktion wirklich in einen bestimmten Block integriert wurde. Wie ist es möglich, dass sie nicht den gesamten Block mit allen Transaktionen herunterladen muss? Der Merkle Tree ist hier die Lösung.

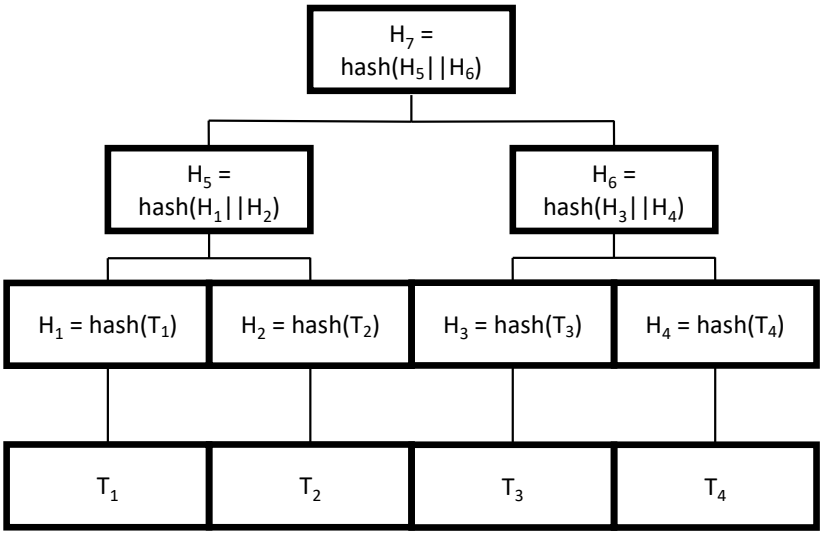

Abbildung 17: Merkle Tree.

[39] Drescher: Blockchain (2017), S.70 - 71.
[40] Tanenbaum/Van Steen: Distributed Systems (2007), S.171 – 172.

Hier sehen wir ein Beispiel für einen Merkle Tree. Unten sind die vier Transaktionen, welche normalerweise auf dem Block gespeichert wären. Die Daten werden gehasht und es entstehen die vier Hashwerte H_1 bis H_4 in der zweiten Zeile. Es werden Paare aus diesen vier Hashwerten gebildet. Hier bilden H_1 und H_2 ein Paar und H_3 und H_4 bilden ein Paar. Die Hashwerte der Paare werden jeweils verkettet (mit dem Symbol «||» dargestellt). Das Verketten bedeutet konkret, dass die Werte aneinandergehängt werden und dann erneut gehasht werden. Somit entstehen die zwei neuen Werte H_5 und H_6 in der dritten Zeile. Diese beiden wieder aneinandergehängt und gehasht und ergeben unseren schlussendlichen Hashwert H_7. Diesen nennt man dann den Merkle Root. Es wird nur dieser Merkle Root in dem Blockheader integriert. Der Blockheader mit dem Merkle Root ist öffentlich und benötigt lediglich 80 Bytes Speicherplatz.

Wenn nun jemand wissen will, ob die Transaktion wirklich in einem Block an einer gewissen Stelle enthalten ist, muss ihm nur ein Teil des Baumes bekannt sein. Dieser Teil des Baumes kann auch aus unehrlichen Quellen erworben werden, da der Merkle Root unveränderbar im Blockheader abgelegt ist. Falls die unehrliche Quelle falsche Informationen bezüglich des gefragten Teils des Baumes gibt, kann dies sofort festgestellt werden. Dann wird beim Berechnen des Merkle Roots ein anderer Wert erhalten. Wegen der Kollisionsfreiheit der Hashfunktionen ist es nicht möglich, einen weiteren abgeänderten Merkle Tree zu finden, welcher denselben Merkle Root hat. Eine Quelle für die fehlenden Teile wäre der Miner des spezifischen Blocks, in dem die Transaktion enthalten sein sollte.

Stellen wir uns nun vor, dass wir sichergehen wollen, dass die erste Transaktion T_1 tatsächlich aussagt, dass Bob uns 5 Bitcoins gesendet hat und diese in einem Block enthalten ist. Um diese zu verifizieren, werden zusätzlich zur Information über die Transaktion lediglich H_2 und H_6 benötigt. Man kann mit genau diesen Werten den Merkle Root ausrechnen und mit dem veröffentlichten Merkle Root vergleichen. Deshalb muss man nicht den ganzen Baum vorliegen haben, sondern nur je einen Wert pro Zeile und die ursprüngliche Transaktion, welche man überprüfen möchte. Das bedeutet, dass die Anzahl der Dateien, welche verschickt werden muss, $\log_2(4) + 1$ ist, da der Logarithmus zur Basis 2 die Anzahl Zeilen angibt. Pro Zeile muss ein Wert vorhanden sein. Die addierte Eins beschreibt die zu überprüfende Transaktion. Die Verifizierung wird somit viel einfacher, da viel weniger Dateien benötigt werden. In unserem Beispiel ist der Unterschied nicht markant, in realen Blockchains können tausende Transaktionen pro Block vorhanden sein, wodurch die Vereinfachung dann sehr viel ausmacht. In den Blockheaders enthalten sind also nur die Merkle Roots. Die einzelnen Teile der Bäume sind nur bei gewissen Computern enthalten, zum Beispiel bei denjenigen, welche die Transaktion tätigen.

Die meisten Merkle Trees funktionieren, wie in Abbildung 17 dargestellt, im Binären System. Das heisst, dass immer Zweierpaare gebildet werden. In unserem Beispiel werden H_1 und H_2 verkettet und dann gehasht und ergeben H_5. Es könnten aber durchaus grössere Gruppen verwendet werden. Es könnten zum Beispiel neun Input-Dateien sein und jeweils die Hashwerte der ersten, zweiten und dritten Datei verkettet und wieder gehasht werden. Unser Beispiel besteht nur aus zwei eigentlichen Zeilen und vier Input Dateien. In realen Anwendungen sind diese Bäume aber sehr viel grösser und haben dementsprechend mehr Inputs. Die Grösse eines Baumes wird jeweils mit der Anzahl der Inputs angegeben. Unser Beispiel hat genau 2^2 Inputs, wobei der Exponent 2 uns genau die Zeilen des Baumes angibt (der schlussendliche Hash H_7 zählt als Zeile 0, da $2^0=1$).

Dieses Verifizierungsverfahren funktioniert, da Hashfunktionen bestimmte Eigenschaften erfüllen. Einerseits kann man nicht auf die ursprüngliche Nachricht zurückschliessen, wenn man nur den Hashwert kennt. Ausserdem ist die Funktion kollisionsresistent, was bedeutet, dass man nicht zwei Inputs findet, welche auf den gleichen Output führen. Dies führt dazu, dass niemand die Transaktionen verändern kann, ohne dabei den Root des Merkle Trees zu

verändern. Das wiederum stellt sicher, dass gefälschte Transaktionen beim Verifizierungsverfahren erkannt werden können. Die Merkle Trees sorgen dafür, dass weniger Daten in den Blöcken enthalten sind. Ausserdem müssen nur wenige Daten vorhanden sein, um Transaktionen zu überprüfen. [41] [42] [43] [44]

3.5. Angriffe auf die Blockchain

3.5.1. Double-Spending Problem

Das Fälschen von Banknoten ist ein schweres Verbrechen, weil die grundlegende Idee von Geld zerstört wird. Man kreiert Macht ohne jegliche Ressourcen dahinter. Daher sind Banknoten mit Sicherheitsmethoden ausgestattet, wie zum Beispiel mit Wasserzeichen, speziellen Materialien oder fluoreszierender Farbe. Diese Sicherheitsmassnahmen machen das Fälschen zwar nicht unmöglich, aber sehr schwierig und kostspielig. Wie sieht die Situation jedoch aus, wenn die Währung digital ist? Ein ähnliches Problem kann in einem Peer-to-Peer Netzwerk mit digitalen Währungen entstehen.

Betrachten wir erneut das Beispiel der Yapis. Yapi Bob will seinen Rai Stone an Yapi Alice weitergeben und verlangt im Gegenzug einen Sack Bananen. Nun müssen alle Yapis auf der Insel über diese Besitzübertragung informiert werden. Sobald alle Yapis über die Besitzübertragung informiert sind, wurde der Rai Stone von Bob an Alice weitergegeben. Während aber diese Information weitergegeben wird, vergeht eine gewisse Zeit. Während dieser Zeit wissen bereits einige von der Besitzübertragung, während andere noch nichts von dieser Besitzübertragung wissen. In dieser Zeitspanne könnte nun Yapi Bob dem Yapi Tom den gleichen Rai Stone anbieten und im Gegenzug erneut einen Sack Bananen verlangen. Damit dieser Tausch funktioniert, darf Tom natürlich noch nichts vom Tausch zwischen Alice und Bob wissen. Nun hat Yapi Bob von Alice und Tom einen Sack Bananen erhalten, dabei aber nur einem Rai Stone ausgegeben. Dieses Problem nennt man das Double Spending Problem und es funktioniert im Zusammenhang mit der Blockchain genau gleich. Während der Verarbeitungszeit einer Transaktion kann eine weitere Transaktion mit demselben Mittel durchgeführt werden. Als Verarbeitungszeit beschreiben wir hier die Zeitspanne, in welcher eine Transaktion vom Pool der unbestätigten Transaktionen in einen Block der Blockchain eingebaut wird. Wie dieses Problem gelöst werden kann, betrachten wir in diesem Abschnitt.

Wir stellen uns hierfür vor, dass Bob analog zum Beispiel der Yapis eine Transaktion A an Alice durchführt, für welche er eine Gegenleistung verlangt. Für diese Transaktion verwendet er 5 Bitcoins, welche er vor einiger Zeit erhalten hat. Gleichzeitig tätigt er eine Transaktion B an Tom. Auch hierfür verlangt er eine Gegenleistung. Verwendet werden für diese Transaktion erneut die 5 Bitcoins, welche er vor einiger Zeit erhalten hat. Beide Transaktionen befinden sich nun im Pool der unbestätigten Transaktionen.

Computer, welche neue Blöcke minen, verwenden hierfür Transaktionen aus dem Pool der unbestätigten Transaktionen. Sowohl Transaktion A als auch Transaktion B werden hierbei als gültig anerkannt. Es ist nicht möglich, dass beide in denselben Block eingebaut werden, da die zweite Transaktion als ungültig gelten würde. Sollten einige Miner nur Transaktion A in ihren neuen Block einbauen, während andere Transaktion B einbauen, existiert eine Version A des Ledgers, in der Transaktion A enthalten ist und eine Version B, in welcher Transaktion B enthalten ist. Werden die beiden Versionen des Ledgers von unterschiedlichen Minern weiterverwendet und durch neue Blöcke ergänzt, wird eine der beiden Ledgers schneller wachsen

[41] The Morpheus Tutorials: Merkle Trees, heruntergeladen am 26.09.2018.
[42] Curran: Merkle Tree, heruntergeladen am 26.09.2018.
[43] Schiller: Merkle Tree, heruntergeladen am 26.09.2018.
[44] Ray: Merkle Tree, heruntergeladen am 14.10.2018

und somit länger werden. Diese Ledger wird bestehen bleiben, da immer nur die längste Version des Ledgers verwendet wird. Eine der Transaktionen A oder B ist deswegen noch nicht in einem Block enthalten und somit erneut im Pool der unbestätigten Transaktionen. Wird diese dann erneut verifiziert um in einen Block eingebaut zu werden, wird dies nicht möglich sein. Grund hierfür ist, dass die 5 Bitcoins schon ausgegeben wurden.

Einen Betrag einer Kryptowährung zweimal auszugeben ist also grundsätzlich nicht möglich. Möglich ist jedoch, dass Bob von Alice und Tom bereits eine Gegenleistung erhalten hat. Dementsprechend wurde entweder Alice oder Tom nicht für ihre Gegenleistung entschädigt.

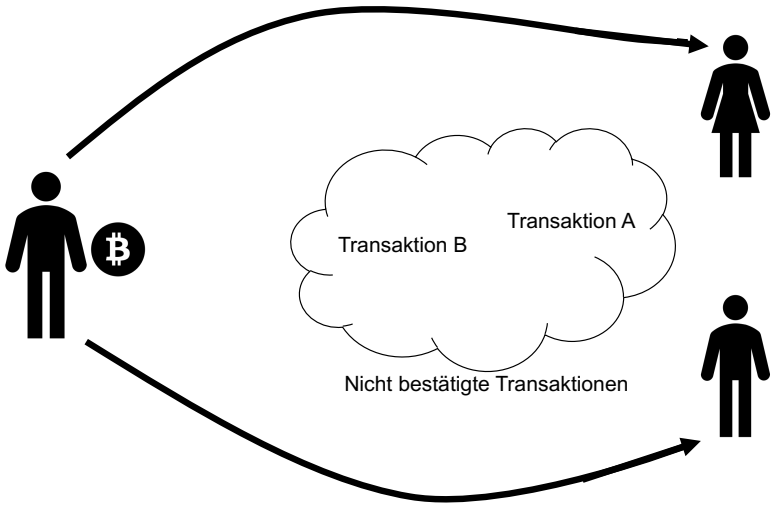

Abbildung 18: Double Spending Problem.

Wenn man als Benutzer sicherstellen will, dass man nicht durch das Double Spending Problem betroffen ist, gibt es eine Lösung hierfür. Stellen wir uns vor Alice möchte ihren Bürostuhl verkaufen. Sie erhält von dem Käufer Bob einen gewissen Betrag an Kryptowährung (beispielsweise ein Bitcoin). Es wäre nun jedoch möglich, dass Bob mit derselben Kryptowährung (also demselben Bitcoin wie vorhin) eine weitere Transaktion bezahlt hat.

Es würden hierbei erneut zwei Versionen A und B des Ledgers entstehen, nur eine davon würde aber schlussendlich überall weiterverbreitet. Alice kann als Benutzer eine Weile warten, bis sie sichergehen kann, dass die Transaktion an sie in der definitiven Version des Ledgers enthalten ist und ihren Bürostuhl dann versenden.

Bei der Bitcoin-Blockchain gilt die Faustregel, dass man etwa sechs neue Blöcke (ca. eine Stunde) warten sollte. Nach dieser Zeitspanne ist es sehr unwahrscheinlich, dass noch immer zwei Versionen des Ledgers im Umlauf sind. Wenn die Transaktion nach einer Stunde immer noch im Ledger vorhanden ist, ist es praktisch sicher, dass Alice ihren Bitcoin erhalten hat. Erst zu diesem Zeitpunkt sollte sie den Bürostuhl verschicken.[45] [46]

[45] Drescher: Blockchain (2017), S.24-26.
[46] Norma Uriarte: Double Spending, heruntergeladen am 09.10.2018.

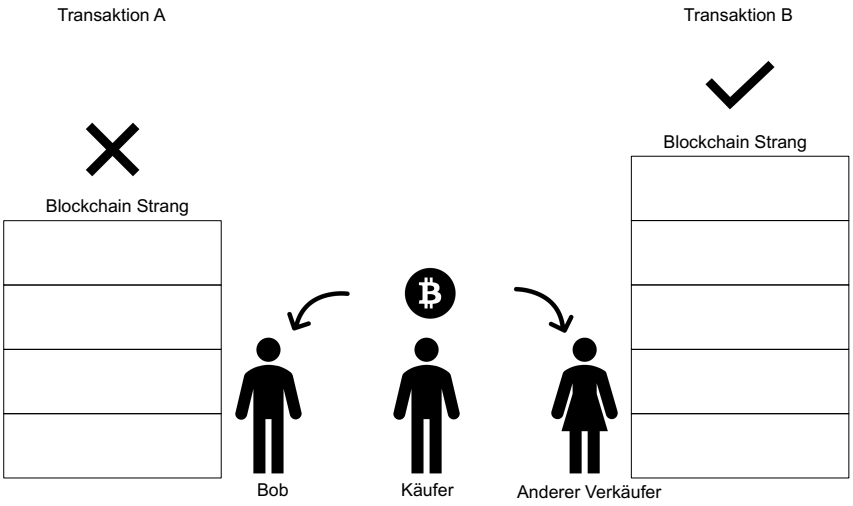

Abbildung 19: Double Spending Problem.

3.5.2. 51% Attacke

Eine weitere Möglichkeit, ein Double Spending Problem zu kreieren, ist die folgende: Besitzt jemand mehr als 50% der Rechenleistung in einem System, ist es dieser Person möglich, den Proof-of-Work schneller zu absolvieren als der Rest des Netzwerks. Diese Person könnte dementsprechend eine Transaktion über beispielsweise einen Bitcoin in einen Block einbauen. Nun könnte diese Person beginnen, eine zweite Version des Ledgers zu kreieren, die nicht verbreitet wird. In dieser zweiten Version ist die erste Transaktion über einen Bitcoin nicht enthalten. Durch die enorme Rechenleistung dieser Person könnte diese zweite Version schnell länger sein als die offizielle Version. Sobald dies der Fall ist, könnte die unehrliche Person ihre Version des Ledgers verteilen. Diese Version würde dann weiterverteilt, da sie die Längste ist. Somit wäre es möglich, ein Double Spending Problem zu provozieren. Dieses Problem wird derzeit heiss diskutiert, da die drei grössten Bitcoin Miner BTC.com, AntPool und ViaBTC zusammen ca. 53% der Rechenleistung der Bitcoin Blockchain besitzen. Wenn sich diese grossen Mining Rechenzentren zusammenschliessen würden, könnten sie eine 51% Attacke durchführen.[47] [48]

3.6. Fragen

9. Was geschieht genau, wenn jemand einen Blockinhalt eines Blocks verändern würde? Was müsste diese Person leisten, damit die Veränderung nicht bemerkt würde und ihre Version des Ledgers trotzdem verbreitet und als die aktuelle Version des Ledgers angesehen würde?
10. Was versteht man unter dem Begriff Mining? Welches sind die Aufgaben der Miner?
11. Wieso gibt es Rechenzentren, welche ausschliesslich zum Minen von Bitcoins existieren? Wieso könnte sich dies lohnen?
12. Ist es theoretisch möglich seinen Computer zum Minen von Bitcoins zur Verfügung zu stellen und dabei einen Gewinn zu erzielen? Lohnte sich dies ebenfalls?
13. Wie wird das Gossip-Prinzip in einem P2P-Netzwerk verwendet?

[47] 51% Attack, in: Investopedia, heruntergeladen am 09.10.2018.
[48] 51% Attack explained, in: mycroptopedia, heruntergeladen am 17.10.2018

14. Welcher Aspekt der Kommunikation in einem P2P-Netzwerk kann mit dem Small Talk unter Menschen verglichen werden?
15. Was ist der Mehrwert des Merkle Trees?
16. Auf einem Block sind 128 Transaktionen gespeichert. Eine Transaktion davon soll sein, dass Alice Bob einen Betrag von 4 Bitcoins überwiesen hat. Wie viele Daten braucht Bob um zu sehen, ob die Transaktion am gewollten Ort vorhanden ist?
17. Wie kann man als Nutzer sicherstellen, dass eine Transaktion wirklich in einem Block integriert wurde und man nicht durch das double spending problem betroffen ist?

4. SMART CONTRACTS

4.1. Funktionsweise von Smart Contracts

Um die Blockchain Technologie besser verstehen zu können, haben wir bisher die einzelnen Elemente der Blockchain an Hand von Geldtransaktionen kennengelernt. Hierbei war das Ziel der Blockchain die ganze Zeit, den sogenannten «Mittelsmann», die Bank, zu umgehen. Zu Beginn haben wir jedoch ebenfalls erfahren, dass die Bank lediglich ein prominentes Beispiel für einen Mittelsmann ist. Es gibt in vielen anderen Gebieten Mittelsmänner, welche mit Hilfe einer Blockchain abgelöst werden könnten. Lediglich eines von vielen Beispielen hierfür sind die Elektrizitätswerke, welche wir ganz zu Beginn erwähnt haben. Doch wie ist es möglich in solch konkreten Beispielen die Blockchain dafür einzusetzen, Mittelsmänner zu ersetzen? Die Lösung hierfür liefern **Smart Contracts**.

4.1.1. Was leistet die Blockchain?

Am konkreten Beispiel von Geldtransaktionen haben wir die Funktionsweise der Blockchain genau betrachtet. Wir wollen hier noch einmal zusammenfassen, was die Blockchain im Allgemeinen genau leistet.

Die Blockchain besteht aus verketteten Blöcken mit einem Inhalt. Dieser Inhalt kann durch das System mit Hashwerten und Nonce nicht verändert werden, ohne dass dies sofort bemerkt wird.

Die Blockchain wird im Peer-to-Peer Netzwerk verbreitet. Dabei wird jeweils immer nur eine Version der Blockchain weitergeleitet. Dies stellt sicher, dass nur eine Version der Blockchain im Umlauf ist, bei welcher sicherlich keine Inhalte manipuliert wurden.

Jedes Handeln von einem Account geht sicherlich vom Inhaber des jeweiligen Accounts aus. Dies garantiert der Digital Signature Algorithm.

4.1.2. Was ist ein Smart Contract?

Bisher waren die Inhalte der Blöcke jeweils Transaktionen. Jedoch ist es auch möglich, statt Transaktionen einen Programmcode auf der Blockchain zu speichern, welcher als **elektronischer Vertrag** dient.

Ein Vertrag ist eine Vereinbarung zwischen zwei oder mehr Parteien. Die Parteien geben hierbei unter bestimmten Voraussetzungen aus freiem Willen ein Angebot ab oder nehmen es an. Ein Kunde in einem Laden legt zum Beispiel seine Zeitung auf die Theke. Damit sagt er stillschweigend aus, dass er bereit ist, einen festgelegten Betrag zu zahlen, um die Zeitung zu erhalten. [49]

Oft sind bei Verträgen Drittpersonen (Mittelsmänner) beteiligt. Ein Beispiel hierfür ist Airbnb. Airbnb stellt den Mittelsmann zwischen Mietern und Vermietern von Wohnungen dar. Mieter zahlen einen Betrag an Airbnb und erwarten als Gegenleistung den Zugang zur gewünschten Wohnung. Vermieter stellen ihre Wohnung zur Verfügung und vertrauen darauf, von Airbnb den Preis für die Wohnungsmiete zu erhalten. Voraussetzung für die Einzahlung des Preises ist hier das zur Verfügung stellen der Wohnung. Hierfür wiederum ist das Einbezahlen des Preises Voraussetzung. [50]

[49] Vertrag, in: business-on, heruntergeladen am 03.10.2018.
[50] Airbnb, in: Wikipedia, heruntergeladen am 03.10.2018.

Der Smart Contract ist eine Möglichkeit, den Mittelsmann (im obigen Beispiel Airbnb) zu umgehen. An diesem Beispiel werden wir zeigen, was das Prinzip eines Smart Contracts ist.

In einem Block auf einer Blockchain wird ein Programmcode gespeichert. Dieser Programmcode heisst Smart Contract. Im Smart Contract ist festgehalten, dass ein Mieter die Wohnung genau dann zur Verfügung gestellt bekommt, wenn er einen festgelegten Preis an den Vermieter zahlt. Der Vermieter erhält sein Geld dann, wenn er die Wohnung tatsächlich zur Verfügung stellt.

Somit muss der Mieter eine Summe an Geld (Kryptowährung) in den Smart Contract einzahlen. Der Vermieter muss seine Wohnung zur Verfügung stellen. Dies kann dadurch möglich gemacht werden, dass der Smart Contract mit dem Türschloss zur Wohnung verbunden ist und diese deshalb öffnen kann. Sind beide Voraussetzungen erfüllt, erfolgt die Übergabe. Der Mieter darf also in die Wohnung einziehen, während der Vermieter seinen Lohn in Form des Geldes erhält. Wird eine Voraussetzung nicht erfüllt, so wird der anderen Partei ihre Leistung zurückerstattet. Der Mieter bekommt also sein Geld zurück oder erhält keinen Zugang zu der Wohnung.

Ein Smart Contract ist prinzipiell ein Vertrag, welcher in Form eines Programmcodes auf der Blockchain gespeichert ist und automatisch ausgeführt wird. Man kann ihn mit Verkaufsautomaten vergleichen. Wenn man eine Münze einwirft, bekommt man die Option, den gewünschten Artikel auszuwählen. Dadurch fällt dieser Artikel dann herunter und man kann ihn entnehmen. Voraussetzungen sind hierbei, dass der Preis des Artikels einbezahlt wird und dass der Artikel vorhanden ist. Der Verkaufsautomat prüft diese Voraussetzungen selbstständig. Sind beide Voraussetzungen erfüllt, so erfolgt die Übergabe des Artikels. Die wichtigste Blockchain bezüglich Smart Contracts nennt sich Ethereum. Ethereum stellt zusätzlich zur Blockchain mehrere Test-Blockchains zur Verfügung, auf denen Smart Contracts ohne den Einsatz von echter Kryptowährung getestet werden können. Die Programmiersprache für Smart Contracts bei Ethereum nennt sich „Solidity“. Sie hat starke Ähnlichkeiten mit der Programmiersprache „Javascript“.

Das wichtigste Prinzip des Smart Contracts ist das sogenannte „If this, then that“-Prinzip. Treffen bestimmte Voraussetzungen zu (if this), so wird eine Aktion ausgeführt (then that). Sowohl das Überprüfen der Voraussetzungen als auch die anschliessende Aktion geschieht automatisch.

Im Beispiel mit Airbnb würde ein Mieter dem Smart Contract möglicherweise über eine Funktion mitteilen, dass er gerne eine Wohnung mieten würde. Der Vermieter würde dem Smart Contract wiederum über eine Funktion mitteilen, ob er damit einverstanden ist. Als erstes würde der Smart Contract nun überprüfen, ob beide Parteien ihr Einverständnis gegeben haben. Ist dies der Fall, fordert der Smart Contract den Mieter darum auf, einen Geldbetrag zu überweisen. Der Smart Contract hat durch das Schloss die Möglichkeit, dem Mieter die Wohnung zur Verfügung zu stellen. Sind nun diese beiden Voraussetzungen gegeben, so gewährleistet der Smart Contract dem Mieter Zutritt und überweist den Geldbetrag an den Vermieter. Diesen Smart Contract könnte man nun noch beliebig detailliert ausbauen. Beispielsweise könnte vom Mieter vor dem Einziehen eine Kaution verlangt werden. Diese wird automatisch zurückbezahlt, wenn keine Gegenstände im Zimmer kaputtgegangen sind. Vorherige Voraussetzung könnte beispielsweise durch eine Kommunikation vom Smart Contract mit einer Kamera und einem Bild- verarbeitungsprogramm funktionieren. Ginge ein Gegenstand kaputt, so könnte der Smart Contract automatisch eine Reparatur durch einen festgelegten Fachmann vereinbaren und diesen dann selbstständig durch die Kaution bezahlen.

4.1.3. Was ist der Vorteil von Smart Contracts?

Doch was ist der Vorteil davon, dass ein solcher Vertrag / ein Programmcode auf der Blockchain festgehalten wird?

Ist ein Smart Contract einmal in einem Block eingefügt, so kann dieser nie mehr verändert werden. Er wird nämlich auf gleiche Weise wie Transaktionen auf der Blockchain festgehalten. Dies verhindert- wie bei den Transaktionen- anschliessende Veränderungen am Programmcode. Wird eine Funktion im Smart Contract aufgerufen, so wird diese auf den Computern im Peer-to-Peer Netzwerk ausgeführt und das Resultat (alle neu definierten Variablen, Ausgabewerte, ...) wird dann gleich wie eine Transaktion erneut in einem Block festgehalten.

Kaum jemand würde einer einfachen Website so weit vertrauen, dass sie einen kompletten Mietvorgang selbstständig ausführt. Der Besitzer der Website könnte durch einfache Änderungen erzielen, dass er alles Geld des Mieters erhält. Hacker könnten Änderungen am Programmcode oder Variablen vornehmen und sich so Zutritt zur Wohnung verschaffen. All dies ist bei Smart Contracts nicht möglich. Sobald sie einmal auf der Blockchain abgespeichert sind, können Smart Contracts von niemandem mehr abgeändert werden. Auch sind die Variablen des Smart Contracts auf der Blockchain abgespeichert und somit nicht veränderbar.

Das Festhalten von Verträgen in Form von Programmcode in der Blockchain hat viele Vorteile. Jedoch ist ein zentraler Nachteil, dass keine Fehler im Programm enthalten sein dürfen. Auch müssen alle Fälle, wie Personen handelt könnten, im Programmcode einbezogen werden. Im Airbnb Smart Contract muss zum Beispiel sichergestellt werden, dass nur der Eigentümer der Wohnung einer Vermietung seiner Wohnung zustimmen kann. Ansonsten könnte jeder diese Funktion aufrufen und somit eine Vermietung bestätigen, welche gegen den Willen des tatsächlichen Vermieters ist.

4.1.4. Beispiele für Smart Contracts

Smart Contracts und ihre Einsatzmöglichkeiten sind sehr vielfältig und interessant. Um dies besser zu verstehen, werden wir uns nun einige konkrete Beispiele ansehen, in denen Smart Contracts schon heute im Einsatz sind.

Energieverteilung

Die Firma HivePower[51] nutzt Smart Contracts zur Energieverteilung. Viele Häuser produzieren schon heute Strom (meist durch Solarzellen), um den eigenen Energiebedarf zu decken. In Zukunft wird dies vermutlich weiterhin stark an Wichtigkeit gewinnen. Zu einigen Zeiten im Tagesverlauf oder Jahresverlauf überschreitet die Eigenproduktion an Energie den Eigenkonsum, zu anderen unterschreitet sie diesen. Momentan ist es so, dass man für den Verkauf der Überproduktion sehr wenig Geld erhält, während der Zukauf von Strom sehr teuer ist. Das meiste Geld geht hierbei an die Energieverteiler. Die Idee von HivePower ist, dass Häuser sich gegenseitig bei dieser Problematik aushelfen. Der Strom wird also direkt zwischen den einzelnen Haushalten verkauft. Dieser Mechanismus wird durch einen Smart Contract geregelt. Er misst automatisch den verkauften beziehungsweise eingekauften Strom der einzelnen Haushalte und regelt die Bezahlung. Dadurch können die teuren Energieverteiler umgangen werden und die Eigenproduktion von Strom wird ausserdem noch attraktiver.

[51] Decentralized Energy Communities, in: hivepower, heruntergeladen am 26.10.2018.

Bauwesen

Während dem Prozess des Baus eines Gebäudes sind eine Vielzahl an Firmen involviert. Einzelne Firmen übernehmen dabei jeweils einzelne Aspekte des Baus (Grundgerüst, Plattenverlegung, Elektrik). Es müssten also viele Verträge abgeschlossen werden, um im Falle von Baumängeln durch Beweise Vertragsverletzungen geltend machen zu können. Beweise sind beispielsweise dann nötig, wenn eine Plattenverlegerfirma den Auftrag bekommt Granitplatten zu verlegen, im Nachhinein jedoch behauptet wird, dass Schieferplatten erwünscht waren. Diese rechtliche Situation wird zusätzlich von ständigen Änderungen in den Bauplänen erschwert.

Durch den Einsatz einer Blockchain im Bauwesen können als Blockinhalte alle Pläne, Abmachungen mit Unternehmen und Änderungen festgehalten werden. Dadurch kann immer nachverfolgt werden, wenn jemand Änderungen an den Bauplänen und Verträgen unternimmt. Auch ist durch den Zeitstempel festgehalten, wann die Änderungen hinzugefügt wurden und es ist ersichtlich, wer Änderungen vorgenommen hat. In den Blöcken der Blockchain ist hier der gesamte Bauprozess festgehalten. Dies erleichtert die Beweislage vor Gericht, da man sich auf die Blockchain beziehen kann, welche wiederum alle wichtigen Informationen enthält.

Transportketten

Ein weiteres Beispiel für eine Anwendung von Smart Contracts bietet die Firma Modum [52]. Ihre Idee besteht darin, die Blockchain Technologie einzusetzen, um die Transportbedingungen verschiedener Medikamente zu verfolgen. Um diese Idee umzusetzen werden Sensoren in die Verpackungen der Produkte eingebaut um beispielsweise Temperatur oder Luftfeuchtigkeit zu protokollieren. Ein Smart Contract liest dann die Messdaten der Sensoren aus und zeichnet den Verlauf der gemessenen Daten auf einer Blockchain auf. Durch den Smart Contract können die Daten somit protokolliert werden, ohne dass sie manipuliert werden könnten. Bei den gemessenen Daten handelt es sich um Bedingungen, die das Medikament oder seine Wirkungen beeinflussen können (z.B. Temperatur oder Luftfeuchtigkeit).

Mit der Idee dieser Firma liessen sich viele weitere Anwendungen umsetzen. Die Sensoren könnten beispielsweise verwendet werden, um Transportwege von Nahrungsmitteln mitzuverfolgen und diese für den Konsumenten auszuweisen.

Zufall in Smart Contracts

Nach obigen Erläuterungen wären Smart Contracts theoretisch perfekt dazu geeignet Glücksspiele zu programmieren. Eine Tatsache verhindert dies jedoch. Wir haben bereits besprochen, dass aufgerufene Funktionen auf mehreren Computern im Netzwerk durchgeführt werden und das Resultat dann in einem Block festgehalten wird. Um Personen Karten zuzuordnen, Würfel zu werfen, Zahlen zu „ziehen“ und Münzen zu werfen werden Zufallsfunktionen benötigt. Diese sollen ein komplett zufälliges Resultat liefern. Würde eine solche Funktion auf mehreren Computern im Netzwerk ausgeführt, so würden sicherlich nicht alle Computer dasselbe Resultat erhalten, welches dann auf der Blockchain abgespeichert werden könnte. Deshalb sind alle „Zufallsfunktionen“ auf der Blockchain keine wirklichen Zufallsfunktionen. Sie liefern meist ein Resultat, welches durch die aktuelle Zeit berechnet wird. So wird das Resultat eines Münzwurfes beispielsweise Kopf, wenn die letzte Ziffer der aktuellen Zeit in Sekunden gerade ist und Zahl, wenn diese Ziffer ungerade ist. Diese Voraussetzungen können

[52] Industry Focused + Cuting-Edge, in: modum, heruntergeladen am 26.10.2018.

jedoch von einem Teilnehmer am Glücksspiel genutzt werden, um den Gewinn für sich einzunehmen. Zufälle in Smart Contracts einzubauen, ist deswegen nicht möglich. [53] [54] [55]

4.2. TicTacToe Contract

Da wir die Smart Contracts als den spannendsten und zukunftsorientiertesten Bereich der Blockchain beurteilen, wurde im Rahmen dieser Maturaarbeit ein eigener Smart Contract in der Programmiersprache Solidity geschrieben. So konnte das Wissen und das Verständnis für diese interessante Anwendung stark vertieft werden.

Der Smart Contract läuft auf der Ethereum Test-Blockchain „Kovan Testnet“. Wir werden diesen Smart Contract im Folgenden genau betrachten und die einzelnen Aspekte davon diskutieren, um ein noch besseres Verständnis für die Blockchain und Smart Contracts zu erlangen.

Das Ziel des entwickelten Smart Contracts ist, ein TicTacToe-Spiel auf der Blockchain umzusetzen. Dabei sollen zwei verschiedene Benutzer mit ihren Accounts gegeneinander spielen. Schlussendlich soll der Sieger einen bestimmten Betrag einer Kryptowährung gewinnen, während der Verlierer diesen Betrag verliert. Dieser Smart Contract wurde in mehrere kleinere Contracts unterteilt, um eine bessere Übersicht zu behalten. Die Contracts „erben“ voneinander. Dies bedeutet in Solidity, dass sie alle Funktionen und Variablen der eingebundenen Contracts übernehmen.

4.2.1. Der Arbeitsprozess

Für das Programmieren eines solchen Smart Contracts musste zuerst die Programmiersprache Solidity erlernt werden. Diese hat Ähnlichkeiten mit bekannteren Programmiersprachen wie beispielsweise Javascript. Dennoch gibt es starke Unterschiede zu dem Programmieren mit herkömmlichen Programmiersprachen. Ein grosser Unterschied ist, dass ein Contract jeweils aus Definitionen von Datenstrukturen wie Variablen, Listen (Arrays) und Funktionen besteht. Die Funktionen werden jedoch nicht nur vom Contract selber aufgerufen, sondern können von ausserhalb oder von einer Funktion im Contract aufgerufen werden. Wir stellen uns eine Funktion namens „add()“ vor. Diese Funktion ist im Contract definiert, sie erhöht eine Variable um eins. Es ist nun nicht möglich, an irgendeiner Stelle des Contracts „add()“ zu schreiben, sodass der Contract die Funktion an dieser Stelle automatisch durchführt. „add()“ könnte entweder von einem Benutzer aufgerufen werden oder von einer anderen Funktion im Contract. Es ist möglich Funktionen als „private“ oder „internal“ zu deklarieren, sie können dann nur vom Contract selbst aufgerufen werden. Dies kann sehr nützlich sein, da einige Funktionen nicht von beliebigen Nutzern aufgerufen werden sollen. Sind die Funktionen öffentlich, werden sie als „public“ deklariert. So kann sie durch den Contract und externe Nutzer aufgerufen werden.

Nach dem Erlernen der Programmiersprache Solidity folgte die Phase, in welcher das TicTacToe Programm geschrieben wurde. Hierbei stand uns Thibault Meunier, welcher bei Liquidity Network - einem Blockchain Startup - als Blockchain Research Engineer arbeitet, bezüglich unserer Fragen zur Verfügung. Besondere Schwierigkeiten stellten sich beim Testen des Programmes. Die beste Entwicklungsumgebung, die bis zum Schluss verwendet wurde, war „Remix“, eine Plattform von Ethereum zum Testen von Smart Contracts. In einer ersten Phase wurde auf einer lokalen Umgebung getestet. Sobald diese Tests keine Fehler mehr aufzeigten, wurde auf einer Blockchain, dem Kovan Testnet, getestet. Die Benutzer-

[53] Blockchain Basics, in medium, heruntergeladen am 03.10.2018.
[54] Hoffmann: Ethereum, Smart Contracts, heruntergeladen am 03.10.2018
[55] Simply Explained – Savjee: Smart Contracts, heruntergeladen am 03.10.2018.

oberfläche von Remix ist auf der folgenden Abbildung zu sehen. Der Programmcode kann in einem Fenster geöffnet werden und dann ausgeführt werden. Sobald der Contract ausgeführt ist, sind in der rechten Spalte die Funktionen und Variablen zu sehen. Über diese lässt sich dann mit dem Contract interagieren.

Abbildung 20: Entwicklungsumgebung Remix

Um auf einer echten Blockchain zu testen, muss man für die Rechenleistung zahlen. Dies wird mit einer Kryptowährung realisiert. Bei Ethereum heissen diese Kosten „Gas“ und werden in der Währung „Ether“ berechnet. Man braucht also Geld um einen Contract auf der Blockchain speichern zu können und um Funktionen daraus aufrufen zu dürfen. Dieses Geld wird in einem „Wallet“ verwaltet. Das meist benutzte Wallet ist MetaMask. Glücklicherweise lassen sich für die verwendete Blockchain Kovan Testnet im Internet kostenlos Ether erwerben, indem man auf bestimmten Plattformen Anfragen verschickt.

Das MetaMask Wallet ist in Abbildung 21 zu sehen. Hierbei wird einerseits das Guthaben sowie die getätigten Transaktionen angezeigt. Auch wird der Status der Transaktionen angezeigt. So sieht man, ob Transaktionen bereits in der Blockchain integriert wurden. Um eine Transaktion zu tätigen, muss man diese zuerst bestätigen, was ebenfalls durch MetaMask geregelt wird.

Die gesamte Blockchain kann auf Etherscan (siehe Abbildung 22) eingesehen werden. Dort kann der Status jeder Transaktion, jedes Contracts und jedes Accounts der Blockchain angezeigt werden.

Sicherheit ist ein zentrales Element, wenn es um Smart Contracts geht. Damit Benutzer besser nachvollziehen können, dass der programmierte Smart Contract sicher ist, wurden zum Schluss zwei weitere Schritte getätigt. Der Smart Contract wurde durch den Sicherheitsscanner Securify.ch getestet, welcher keine für diesen Contract kritischen Sicherheitsfehler erkannte. Weiterhin wurde der Code auf Etherscan hochgeladen und verifiziert, sodass Benutzer diesen ansehen und selber überprüfen können, dass der Code keine Sicherheitsfehler aufweist.

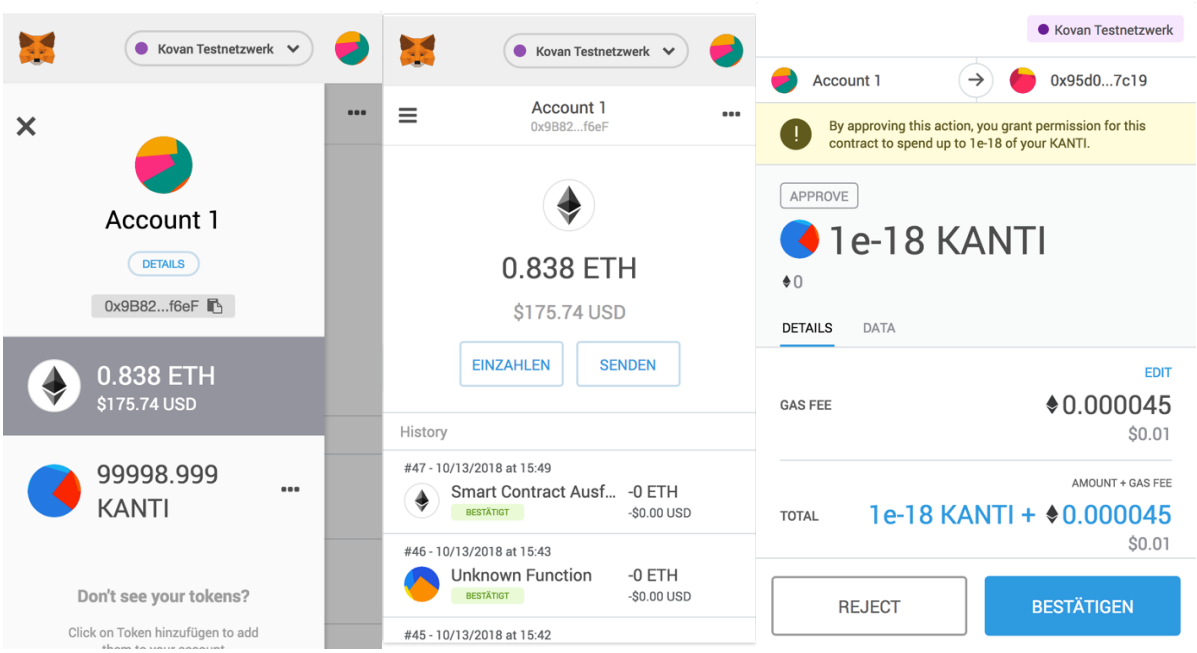

Abbildung 21: Metamask Wallet

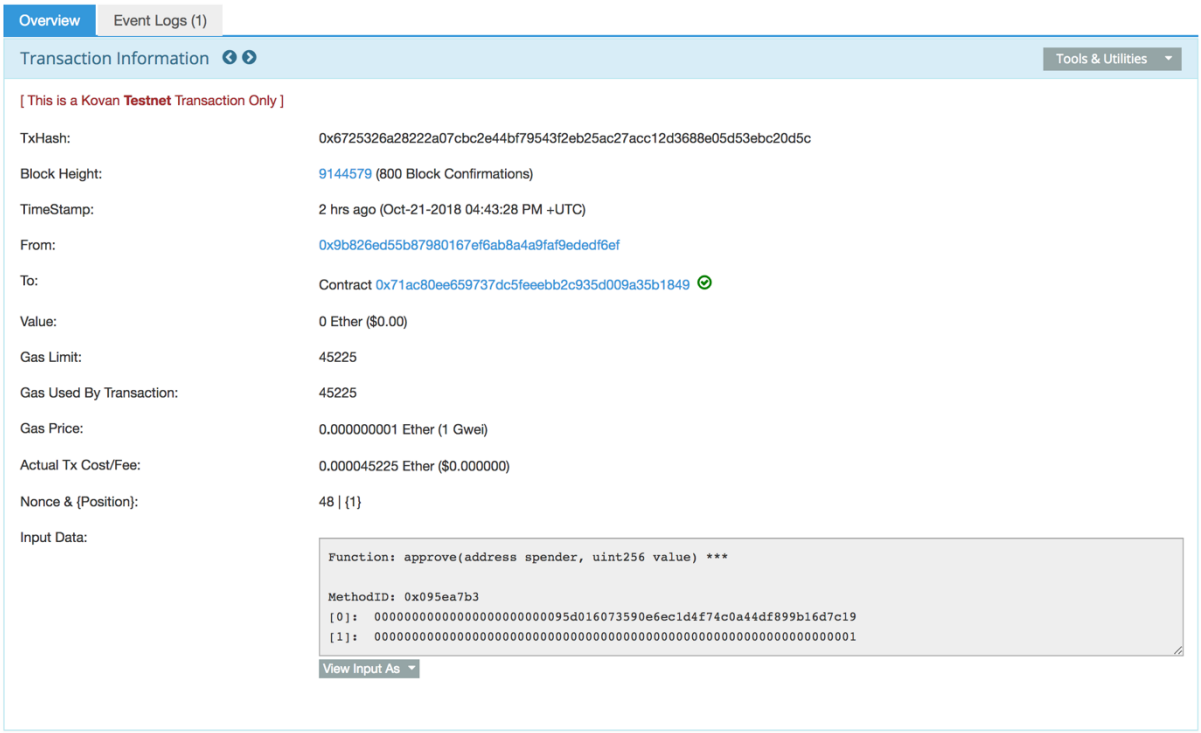

Abbildung 22: Etherscan

Die Transaktion wird jeweils genauer beschrieben, beispielsweise durch den Hashwert der Transaktion und den Input. In diesem spezifischen Fall wurde eine Funktion namens „approve“, welche wir später nochmals antreffen werden, auf der Blockchain ausgeführt. Diese Durchführung wurde vom Account (0x9B826ed55b87980167Ef6ab8A4A9fAf9eDedf6eF) getätigt. Als Empfänger ist hier ein Contract angegeben, da die Funktion aus diesem Contract stammt. Adresse des Contracts ist hier 0x71ac80ee659737dc5feeebb2c935d009a35b1849.

Im Folgenden sind die einzelnen Contracts unseres TicTacToe Beispiels beschrieben.

4.2.2. userDefinition

Zuerst werden zwei Adressen erstellt, welche „PlayerA“ und „PlayerB“ heissen. Adressen sind, wie wir beim DSA-Algorithmus erfahren haben, öffentliche Schlüssel. Die beiden Adressen werden schlussendlich die zwei gegnerischen Spieler sein. Standardmässig werden sie jedoch momentan lediglich auf 0 gesetzt. Ausserdem wird im sogenannten „mapping“ darüber informiert, dass diesen Adressen später etwas zugeordnet werden wird, nämlich eine Zahl. Diese Zuordnung nennen wir hier „playerSymbol“. Die Variable „playerDefined“ beschreibt, wie viele Spieler bereits am Spiel teilnehmen. Anfangs sind dies keine (also Null) Spieler. Maximal dürfen es zwei sein. Eine letzte Variable namens „endGame“ wird hinzugefügt. Sie kann die beiden Werte „true“ oder „false“ annehmen. Ist „endGame“ wahr, so ist das Spiel bereits beendet, ist sie falsch, so findet gerade ein Spiel statt. Standardmässig wird diese Variable zuerst als wahr definiert.

```
address internal PlayerA;
address internal PlayerB;
mapping (address => uint8) internal playerSymbol;
uint8 internal playerDefined = 0;
bool internal endGame = true; //ist diese Variable True, so hat das Spiel bereits
geendet
```

Als nächstes wird eine Funktion definiert, die wir „defineUsers“ nennen. Sie ist öffentlich, jeder kann dementsprechend diese Funktion aufrufen. Zuerst wird verlangt, dass das vorhergehende Spiel bereits beendet wurde (oder dies das erste Spiel ist). Wäre dies nicht der Fall, so darf diese Funktion nicht ausgeführt werden. Schliesslich könnte ein unehrlicher Spieler sonst, wenn er dabei ist zu verlieren, ein neues Spiel beginnen um seiner Niederlage zu entgehen.

Ist „PlayerA“ noch nicht definiert (beträgt „PlayerA“ also momentan 0), so wird der Account, welcher die Funktion momentan aufruft (msg.sender) als „PlayerA“ definiert. „playerDefined“ wird dementsprechend auf 1 gesetzt. Ausserdem bekommen alle Spielfelder (fields) den Wert Null. Dies ist wichtig, falls vor diesem Spiel bereits eines stattgefunden hat. Die Spielfelder werden im nächsten Contract definiert. Dann werden wir diesen Teil besser verstehen.

Ist „PlayerA“ bereits definiert so wird überprüft, dass derjenige, der die Funktion aufruft nicht „PlayerA“ ist. Ist dies wiederrum der Fall, so wird derjenige, der die Funktion aufgerufen hat, als „PlayerB“ definiert. „PlayerA“ erhält das „playerSymbol“ 1, „PlayerB“ erhält das „playerSymbol“ 2.

Schlussendlich wird die „endGame“ Variable auf „false“ gesetzt, das Spiel beginnt jetzt also. Diese Variable ist dafür zuständig, dass niemand mehr die Funktion „defineUsers“ aufrufen kann, die Anmeldephase nun geschlossen ist. Ohne sie könnte jemand, wie bereits oben beschrieben, zu einem beliebigen Zeitpunkt das Spiel abbrechen und ein neues Spiel beginnen, was verhindert werden soll.

```
function defineUsers() public {
    require(endGame == true);
    if (PlayerA == 0x0000000000000000000000000000000000000000) {
        PlayerA = msg.sender;
        playerSymbol[PlayerA] = 1;
        playerDefined = 1;
        for (uint8 i=0; i<=2; i++) {
            for (uint8 k=0; k<=2; k++) {
                fields[i][k] = 0;
```

```
            }
        }
    }
    else if (PlayerA != msg.sender) {
        PlayerB = msg.sender;
        playerSymbol[PlayerB] = 2;
        playerDefined = 2;
        endGame = false;
    }
}
```

Es werden zwei weitere Funktionen definiert, welche ermöglichen nachzusehen, wer „PlayerA“ ist und wer „PlayerB“ ist. Sie haben als Ausgabewert jeweils die Adressen von „PlayerA“ und „PlayerB“. Diese Funktionen wurden vor allem für Testzwecke benötigt.

```
function getPlayerA() view public returns (address) {
    return PlayerA;
}
function getPlayerB() view public returns (address) {
    return PlayerB;
}
```

4.2.3. fields

Der nächste Contract heisst „Fields“. In ihm wird der Aufbau des Spielfelds geregelt. Zuerst wird ein 2-dimensionales Array kreiert. Das bedeutet eine Matrix von Werten (hier Zahlen) in Zeilen und Spalten. Standartmässig sieht dieses folgendermassen aus:

[0,0,0]

[0,0,0]

[0,0,0]

Hierbei wird die erste Zeile jeweils als 0. Zeile bezeichnet, der erste Eintrag wird als 0. Eintrag bezeichnet.

```
uint8 [3][3] internal fields;
```

Als nächstes wird eine Funktion definiert, die ermöglicht, den Wert eines Feldes zu wechseln. Die Funktion nimmt als Eingabewert die Zeile und Spalte des Feldes und den gewünschten Wert. Hierbei kann jeweils die 1. Zeile als Zeile 1 bezeichnet werden und muss nicht als Zeile 0 bezeichnet werden. Die Funktion übernimmt die Anpassung selber. Diese Funktion ist nicht öffentlich, sondern intern, da wir sie später in einem weiteren Contract verwenden, sie jedoch niemand ausser der Contract selbst aufrufen darf. Schliesslich könnte jemand ansonsten die Werte aller Felder beliebig ändern.

```
function fieldChange (uint8 i, uint8 k, uint8 value) internal {
    fields[i-1][k-1] = value;
}
```

Die Funktion „getFieldValue“ ermöglicht es den Wert eines Feldes anzusehen. Die Funktion „getAllFieldValues“ ermöglicht es, die Werte aller Felder anzusehen. Dabei wird eine neunstellige Zahl kreiert, welche an der i. Stelle den Wert des 10-i. Feldes beschreibt. Als das erste Feld bezeichnen wir hier das Feld links oben, als zweites Feld bezeichnen wir das Feld, welches sich oben in der Mitte befindet, und so weiter. Die Funktion „getAllFieldValues“ wird hauptsächlich für einen späteren Schritt benötigt.

```
function getFieldValue(uint8 i, uint8 k) view public returns (uint8) {
    return fields[i-1][k-1];
}
function getAllFieldValues() view public returns (uint256) {
    uint8 i;
    uint8 k;
    uint256 result=0;
    uint256 position=0;
    for (i=0; i<=2; i++) {
        for (k=0; k<=2; k++) {
            result = result + (fields[i][k] * (10**position));
            position++;
        }
    }
    return result;
}
```

4.2.4. winner

Dieser Contract dient dazu zu überprüfen, ob jemand gewonnen hat. Er enthält jeweils Funktionen, welche überprüfen, ob in der Zeile und Spalte, in der zuletzt gespielt wurde, drei gleiche Symbole sind. Auch die Diagonalen werden überprüft.

Die untenstehende Funktion „wonGameColumnk“ überprüft, ob sich in der Spalte k dreimal der Wert „playerValues“ befindet. Es werden alle Werte der Spalte k durchgegangen. Beträgt ein Wert „playerValue“, so wird eine Variable „count“ um eins erhöht. Beträgt diese Variable am Schluss 3, so ist der Ausgabewert der Funktion „true“, anderenfalls ist er „false“. Im Contract steht eine ähnliche Funktion, welche dasselbe für die Spalte i ausführt. „true“ bedeutet, dass der Spieler, welcher den „playerValue“ als Symbol besitzt, 3 seiner Symbole in einer Spalte hat und somit Sieger ist.

Eine fast identische Funktion für die Zeilen ist ebenfalls in diesem Contract erhalten, hier jedoch nicht noch einmal aufgeführt.

```
function wonGameColumnk (uint8 playerValue, uint8 column) internal returns (bool)
{
    count = 0;
    for (uint8 i=1; i<=3; i++) {
        if (fields[i-1][column-1] == playerValue) {
            count++;
        }
    }
    if (count == 3) {
        return true;
    }
    else {
        return false;
    }
}
```

Die Funktion „wonGameDiagonal“ überprüft, ob sich in den beiden Diagonalen dreimal der Wert „playerValue“ befindet.

Dies auf analoge Weise wie bei der Funktion für die Spalte.

```
function wonGameDiagonal (uint8 playerValue) internal returns (bool) {
//von unten links nach oben rechts
    count = 0;
    uint8 i = 3;
    for (uint8 k=0; k<=2; k++) {
        i = i - 1;
        if (fields[i][k] == playerValue) {
           count++;
        }
    }

    if (count == 3) {
        return true;
    }
//von oben links nach unten rechts
    count = 0;
    i = 0;
    for (k=0; k<=2; k++) {
        if (fields[i][k] == playerValue) {
            count++;
        }
        i++;
    }

    if (count == 3) {
        return true;
    }
    else {
        return false;
    }
}
```

4.2.5. playGame

Der nächste und letzte Teil regelt den Ablauf des Spieles und das Spielende. Zuerst werden einige Variablen definiert. „whichPlayer“ gibt an, wer an der Reihe ist. 1 bedeutet, dass PlayerA an der Reihe ist, 2 bedeutet, dass „PlayerB“ an der Reihe ist. Anfangs ist „whichPlayer“ 1, da „PlayerA“ beginnt.

„moveCount“ zählt die Anzahl Züge, dies dient dazu zu entscheiden, wenn ein Spiel unentschieden ist. „helpWinner“ beträgt momentan 0. Später dient diese Variable dazu zu zeigen, wer gewonnen hat.

```
uint8 internal whichPlayer = 1;
uint8 internal moveCount = 0;
uint8 internal helpWinner = 0;
```

Nun werden einige „events“ definiert. „events“ sind Mitteilungen, an jemanden, der gerade eine Funktion aufruft. Diese „events“ werden in späteren Funktionen benötigt. „gameOver“

teilt mit, wer gewonnen hat und eine zusätzliche Nachricht wird mitgeteilt. „Error“ dient dazu jemandem eine Nachricht mitzuteilen, wenn er einen Fehler gemacht hat.

```
event gameOver(
    address Winner,
    string message
);
event Error(string mistake);
```

Die nächsten Funktionen dienen dazu, Informationen über das Spiel zu erlangen. „statusInit“ zeigt an, wie viele Spieler momentan im Spiel sind. „checkWinner“ zeigt den Sieger an, falls bereits jemand gewonnen hat. „whoAmI“ dient dazu herauszufinden, welcher Spieler man ist. Die Funktion „whoseTurn“ gibt als Ausgabewert denjenigen Spieler an, welcher gerade an der Reihe ist. Dies wird mit Hilfe von „whichPlayer“ erreicht. Ist das Spiel fertig, so ist der Ausgabewert dieser Funktion der Sieger der Partie.

```
function statusInit() view public returns (uint8) {
    return playerDefined;
}
function checkWinner () view public returns (uint8) {
    if (endGame) {
       return helpWinner;
    }
    else {
       return 0;
    }
}
function whoAmI() view public returns (uint8) {
    if (PlayerA == msg.sender) {
        return 1;
    }
    else if (PlayerB == msg.sender) {
        return 2;
    }
    else{
        return 0;
    }
}
function whoseTurn() view public returns (string) {
    if (endGame) {
        return helpWinner;
    }
    else if (whichPlayer == 1) {
        return "PlayerA";
    }
    else {
       return "PlayerB";
    }
}
```

Eine weitere wichtige Funktion ist die „Reset“-Funktion. Ist das Spiel beendet, so wird sie aufgerufen, um alle Werte so zurückzusetzen, dass ein neues Spiel gestartet werden kann.

„helpWinner“ wird nicht auf 0 gesetzt, damit der Sieger der vorherigen Runde abgelesen werden kann.

```
function reset() internal {
    endGame = true;
    playerDefined = 0;
    PlayerA = 0x0000000000000000000000000000000000000000;
    PlayerB = 0x0000000000000000000000000000000000000000;
    whichPlayer = 1;
    moveCount = 0;
}
```

Die nächste Funktion „Play“ ist diejenige Funktion, die aufgerufen wird, um einen Spielzug zu tätigen. Sie ist öffentlich, da sie von den Spielern des Spiels aufgerufen werden soll, um ihr Symbol auf ein Feld zu setzen. Als Eingabewert werden die Zeile und Spalte des gewünschten Feldes verwendet. Zuerst wird die Variable „playerValue“ definiert als das „playerSymbol“ von dem, der die Funktion aufruft.

Ist das Symbol desjenigen, der die Funktion aufruft gleich der Variablen „whichPlayer“, so ist dieser Spieler aktuell an der Reihe. Ist zusätzlich hierzu der aktuelle Wert des gewählten Feldes 0 (das Feld wurde noch nicht besetzt), so wird das Symbol des Spielers in dieses Feld gesetzt. Ausserdem wird der „moveCount“ um eins erhöht. Nun wird überprüft, ob der letzte Spielzug zu einem Sieg des aktuellen Spielers geführt hat. Ist dies der Fall, so wird eine Funktion „Winner“ aufgerufen, die wir uns nachher gleich genauer ansehen. Zudem wird nachgesehen, ob das Spiel unentschieden geendet hat (der „moveCount“ = 9 ist). Ist dies wiederrum der Fall, so wird die Funktion „winnerNobody“ aufgerufen, welche wir wiederum nachher genau ansehen.

War das Feld schon besetzt, so geschieht nichts, lediglich die Nachricht „field is already occupied“ wird gesendet. Ist derjenige, der die Funktion aufgerufen hat nicht an der Reihe, wird wiederum nur eine Nachricht gesendet: „it’s not your turn yet."

```
function Play(uint8 i, uint8 k) public {
    uint8 playerValue = playerSymbol[msg.sender];
    if (whichPlayer == playerValue) {
        if (fields[i-1][k-1] == 0) {
            fieldChange (i, k, playerSymbol[msg.sender]);
            if (msg.sender == PlayerA) {
                whichPlayer = 2;
            }
            else {
                whichPlayer = 1;
            }
            moveCount ++; //PlayerB is allowed to play now
            //checking if PlayerA has won:
            if (wonGameColumnk(playerValue, k) == true) {
                winner(msg.sender);
            }
            if (wonGameRowi(playerValue, i) == true) {
                winner(msg.sender);
            }
            if (wonGameDiagonal(playerValue) == true) {
                winner(msg.sender);
            }
            if (moveCount == 9) {
                winnerNobody();
```

```
            }
        }
        else {
            emit Error("field is already occupied");
        }
    }
    else {
       emit Error("it's not your turn yet");
    }
}
```

Die vorher verwendete Funktion „winner“ nimmt als Eingabewert eine Adresse (den Gewinner). Es wird eine Nachricht gesendet, dass diese Adresse das Spiel gewonnen hat. Ist der Gewinner (die Adresse) gleich der Adresse von Player A so wird helpWinner zu 1, anderenfalls (wenn Player B gewonnen hat) wird „helpWinner“ als 2 definiert. Die „Reset“ Funktion wird aufgerufen.

```
function winner(address checkWinnerId) internal {
    emit gameOver(checkWinnerId, "has won the Game");
    if (checkWinnerId == PlayerA) {
        helpWinner = 1;
    }
    else {
       helpWinner = 2;
    }
    reset();
}
```

Die Funktion „winnerNobody“ wird aufgerufen, wenn das Spiel unentschieden ausgeht. „helpWinner“ wird dann als „no winner!“ definiert. „endGame“ wird erneut auf wahr gesetzt.

```
function winnerNobody() internal {
    helpWinner = 3;
    reset();
}
```

4.3. ERC20 Contract

Mit der obigen Funktionalität ist das Spiel bereits spielfähig. Wir haben oben jedoch erwähnt, dass ein zusätzliches Ziel des Spieles ist, dass der Sieger der jeweiligen Partie einen bestimmten Betrag gewinnt. Im Rahmen der Maturaarbeit wurde hierfür eine eigene Kryptowährung kreiert. Dabei unterscheidet man die eigentliche Kryptowährung der Blockchain (bei Ethereum Ether) und Tokens. Tokens sind wiederum Smart Contracts, welche als „digitale Wertgegenstand“ genutzt werden können. Zum Kreieren von diesen Tokens stellt Ethereum einige Standards zur Verfügung. Diese Token-Standards sind Vorlagen für die Erstellung von Tokens und den Transfer von einem Account auf einen anderen. Das ist wichtig, da Fehler schnell zum Verlust von grossen Werten führen können.

4.3.1. ERC20 Token-Standard

Einer dieser Standards ist der ERC20-Token-Standard. Er ist dazu geeignet Tokens zu kreieren, bei welchen jedes Token identisch zu allen anderen sein soll, also wie eine Kryptowäh-

rung. Man könnte diesen „digitalen Wertgegenstand“ mit Geld vergleichen (das jedoch nur virtuell und nicht materiell existiert).

Ein Beispiel für Tokens mit anderen Eigenschaften sind die Tokens des ERC721-Token Standards. Hierbei handelt es sich um Tokens, welche unterschiedliche Eigenschaften haben können. Diese könnte man beispielsweise mit Ohrringen, Uhren oder Kleidern vergleichen. Ein wichtiges Beispiel für ERC721-Tokens sind Cryptokitties. Es handelt sich hierbei um einen Smart Contract in welchem (digitale) Kätzchen mit bestimmten Eigenschaften eingekauft und verkauft werden. Die Kätzchen sind hierbei ein ERC721-Token. Obwohl die Cryptokitties lediglich digital existieren, geben einige Leute über 100’000 Dollar für die Cryptokitties aus. [56] [57]

4.3.2. ERC20 Interface

Für unseren Contract brauchen wir eine Kryptowährung, verwenden also einen Smart Contract des ERC20-Token Standards. Dieser verfügt über eine Vielzahl von Funktionen. Die wichtigsten hiervon sind: „TokenERC20“, „balanceOf“, „transferFrom“, „approve“ und „transfer“. „TokenERC20“ ist eine „constructor“-Funktion. Sie wird nur einmal beim Bereitstellen des Contracts auf einer Blockchain aufgerufen. In diese Funktion werden die Parameter des eigenen Tokens eingesetzt. Hierbei muss bestimmt werden, wie viele Tokens es insgesamt geben soll, wie der Name der Tokens sein soll und wie das Symbol der Tokens sein soll. „balanceOf“ ist eine Zuordnung, welche dazu dient festzustellen, über wie viele Tokens eine Adresse verfügt. Die „transferFrom“ Funktion überweist eine festgelegte Menge an Tokens von einem Account auf einen weiteren Account. Hierbei darf nicht auf den Account 0x0 überwiesen werden, da dieser nicht existiert und somit Tokens verloren gehen würden. Dieses Verlorengehen wird meist als „Verbrennen von Tokens“ bezeichnet. Bei der „transferFrom“-Funktion muss der Besitzer des Accounts, von dem überwiesen wird jeweils einwilligen, damit diese Funktion ausgeführt werden kann. Dieses Einwilligen geschieht durch die „approve“-Funktion. Diese Funktion wird vorgängig aufgerufen um jemandem (einer Adresse) zu erlauben, eine gewisse Menge des Tokens von sich zu überweisen. Die „transfer“-Funktion überweist eine festgelegte Menge an Kryptotokens von dem aktuellen Contract (in welchem die Funktion aufgerufen wird) an einen bestimmten Account. [58]

Das Token, welches im Rahmen der Maturaarbeit erstellt wurde heisst „KantiCoin“ mit dem Symbol „KANTI“. Wie dieses nun in den Contract eingebunden wird, werden wir als nächstes besprechen.

Zuerst muss ein Interface zwischen unseren bisherigen Contracts und dem Kanti-Coin Contract hergestellt werden. Unser Contract muss also mit dem Kanti-Coin Contract interagieren können. Dieses Interface lässt sich über die Adresse des Kanti-Coin Contracts auf Kovan Test Net herstellen.

Zu Beginn des Spiels, also beim Aufrufen der „defineUser“ Funktion, müssen beide Spieler 0.1 KANTI an den Contract überweisen. Diese Überweisung wird durch die „transferFrom“ Funktion ausgeführt. Bedingung ist dementsprechend, dass der Spieler dem Contract durch die „approve“ Funktion das Recht gegeben hat, 0.1 KANTIs des jeweiligen Spielers an sich zu überweisen. Die Transaktion muss dringend vor Spielbeginn geschehen. Würde diese Transaktion am Ende direkt vom Verlierer an den Gewinner stattfinden, so könnte der Verlierer diese Transaktion einfach ablehnen und der Sieger würde seine Belohnung nicht erhalten.

[56] What is a Cryptocurrency Token?, in: CryptoCurrency Facts, heruntergeladen am 11.10.2018.
[57] Blockchain: Digital Cats, in: Fortune, heruntergeladen am 11.10.2018.
[58] Crypto-currency, in: Ethereum, heruntergeladen am 12.07.2018

Dadurch, dass die Transaktion vor Spielbeginn eingefordert wird, müssen die Spieler ihr zustimmen, anderenfalls könnten sie nicht spielen.

Endet das Spiel unentschieden, so bekommen beide Spieler ihre 0.1 KANTI vom Contract zurück. Gibt es am Schluss einen Sieger, so erhält dieser vom Contract 0.2 KANTI, während der Verlierer nichts erhält. Diese Transaktion wiederum wird durch die „transfer“ Funktion durchgeführt.

Eine weitere mögliche Implementierung wäre gewesen, dass bei einem unentschiedenen Spiel keiner der Spieler KANTIs zurückbekommt. Die Folge hiervon wäre, dass der Contract zu einem Jackpot wird. Der nächste Sieger bekäme dann die gesamte Summe des Jackpots. Hierbei entstünde jedoch ein Problem, beziehungsweise eine Möglichkeit für unehrliche Spieler unberechtigt an KANTIs zu kommen. Wenn Spiele unentschieden enden würden, so gäbe es die Möglichkeit, dass eine Person sich mit 2 Accounts anmeldet. Diese Person könnte dann so gegen sich selbst spielen, dass sie mit einem Account verliert und mit dem anderen gewinnt. So würde sie ohne Verluste zu machen die gesamte Summe des Jackpots gewinnen.

4.3.3. Zeitstempel

Bemerkte ein Spieler, dass er dabei ist zu verlieren, so könnte er einfach aufhören zu spielen. Dies würde dazu führen, dass sein Gegner keinen Gewinn erhielte. Ausserdem wäre das Spiel blockiert, neue Spieler könnten also nicht mehr beitreten. Hierfür muss eine Lösung gefunden werden. Diese ist eine Funktion namens „checkTime“. Zuerst wird eine Variable „playTime“ als Null definiert. Macht ein Spieler einen Spielzug, so wird „playTime“ auf exakt die Zeit, zu der er gespielt hat, definiert. In der Funktion „checkTime“ wird die aktuelle Zeit mit „playTime“ verglichen. Ist die aktuelle Zeit über einen Tag nach der „playTime“, so verliert der Spieler, welcher nicht gespielt hat, automatisch. Diese Funktion kann von den beiden Spielern aufgerufen werden. Ausserdem wird sie jedes Mal aufgerufen, wenn ein Spieler neu ins Spiel einsteigen will. Wenn keine neuen Spieler das Spiel beginnen wollen und keiner der beiden Spieler weiterspielt, so ist die Blockade kein Problem, da zu diesem Zeitpunkt niemand spielen will.

4.4. Web3js Schnittstelle

Der oben beschriebene TicTacToe Contract ist der Hauptteil unserer Arbeit. Er ist in der dargelegten Form bereits komplett funktionsfähig und kann verwendet werden, um damit TicTacToe zu spielen. Dies wird, wie oben bereits erklärt wurde, durch Remix ermöglicht. Einige Aspekte sind jedoch sehr mühsam, wenn über Remix gespielt wird. Erstens muss allen Spielteilnehmern die Adresse des Contracts auf der Blockchain und der komplette Programmcode bekannt sein, um den Contract in Remix einzubinden und Funktionen in diesem Contract aufzurufen. Zusätzlich müssen die Nutzer für sich selbst ein Spielfeld zeichnen, um den Überblick zu behalten. Zuletzt muss jeder Spieler selbstständig durch Aufrufen einer Funktion überprüfen, wer gerade an der Reihe ist.

Deswegen entschlossen wir uns dazu, zusätzlich zum Contract eine Website zu erstellen, welche mit dem Contract zusammenarbeitet. Der Webentwickler Wojciech Kaluzny hat eine Website entworfen und ein Javascript TicTacToe Spiel darauf implementiert. Mit dieser Anwendung können Spieler an ihrem Rechner lokal TicTacToe gegen einen Computer spielen. Seinen Code für diese Website hat er öffentlich zur Verfügung gestellt (Open Source), sodass wir diesen nutzen konnten und ihn als Grundlage für das Design unserer Website verwenden konnten.[59]

[59] Kaluzny: TicTacToe, heruntergeladen am 29.09.2018

Wir verwendeten aus seinem Programm die Darstellung des Spielfeldes und das Aufrufen von Funktionen beim Klicken in ein Feld für unsere Website weiter. Als nächster Schritt musste erreicht werden, dass auf der Website Funktionen der Blockchain aufgerufen werden können. Die Möglichkeit hierfür basiert auf dem Programm web3.js, welches eingebunden in eine Website ermöglicht, dass eine Schnittstelle zwischen dieser Website und einer Blockchain hergestellt werden kann. Metamask ermöglicht diese Verbindung zur Blockchain und verwaltet zusätzlich als Wallet die Transaktionen der Benutzer.

Die gesamte Schnittstelle wird über ein Javascript Programm hergestellt. Um auf den Contract zuzugreifen werden zwei Informationen über den Contract benötigt: Die Adresse und das ABI (Application Binary Interface). Das ABI eines Contracts dient als Beschreibung der Schnittstelle des Contracts für aufrufende Programme. Beide Informationen können beim Speichern des Contracts auf der Blockchain aus Remix angesehen und kopiert werden. Danach werden sie im Javascript-Programm gespeichert. Mit diesen beiden Informationen ist die Schnittstelle zu dem Smart Contract im Javascript-Programm definiert und die Funktionen aus genau diesem Smart Contract können aufgerufen werden.

In unserer Website wurde sowohl eine Schnittstelle zum TicTacToe Spiel als auch eine zu unserem ERC20 Contract definiert. Im Folgenden wird der Programmablauf beschrieben. Als erstes wird „statusInit“ benötigt, um auf der Website darzustellen, wie viele Spieler bereits definiert sind. Als nächstes wird die „approve“ Funktion aus dem ERC20 Contract verwendet. Ein Button wird so definiert, dass beim Anklicken des Buttons die „approve“ Funktion aufgerufen wird. Der zweite Button ruft beim Anklicken die „defineUser“ Funktion auf. Sobald ein Benutzer die Transaktion freigegeben hat („approve“), kann er dem Spiel beitreten indem er den „defineUser“ Button wählt. Sobald zwei Spieler beigetreten sind, beginnt das Spiel. Die Website zeigt mit Hilfe der „whoseTurn“ Funktion an, wer an der Reihe ist. Mit der „checkWinner“ Funktion wird regelmässig überprüft, ob jemand gewonnen hat. Das Spielen ist möglich, indem ein Feld von demjenigen Spieler, der an der Reihe ist, angewählt wird. Beim Klicken auf das entsprechende Feld wird nämlich die „Play“ Funktion aus dem TicTacToe Contract aufgerufen.

Damit der aktuelle Stand des Spielfeldes angezeigt wird, müssen die Werte aller Felder regelmässig, bei uns alle 10 Sekunden, neu geladen werden. Dies wird erreicht, indem die „getAllFieldValues“ Funktion aus dem TicTacToe Contract aufgerufen wird. Die Werte der neunstelligen Zahl werden in 9 einzelne Werte zerlegt, so dass jedem Feld der korrekte Wert zugeteilt und angezeigt wird. Statt null, eins oder zwei werden auf der Website jeweils ein leeres Feld, ein Kreuz oder ein Kreis angezeigt.

Wird eine Variable im Smart Contract verändert, so muss dies in einem neuen Block in der Blockchain dokumentiert werden. Bis diese Veränderung auf der Blockchain festgehalten ist, vergeht eine Zeit. In dieser Zeit können keine weiteren Funktionen aus dem jeweiligen Smart Contract aufgerufen werden. Deswegen wird nach Aufrufen jeder Funktion, solange gewartet, bis die Änderung in der Blockchain festgehalten wurde. Dies hat den Nutzen, dass keine Verwirrungen bei den Spielern entstehen. Eine spezielle Funktion aus dem web3.js ermöglicht festzustellen, ob eine Transaktion bereits in einem Block aus der Blockchain festgehalten wurde. [60]

[60] Ethereum Contract ABI, in: github, heruntergeladen am 26.10.2018.

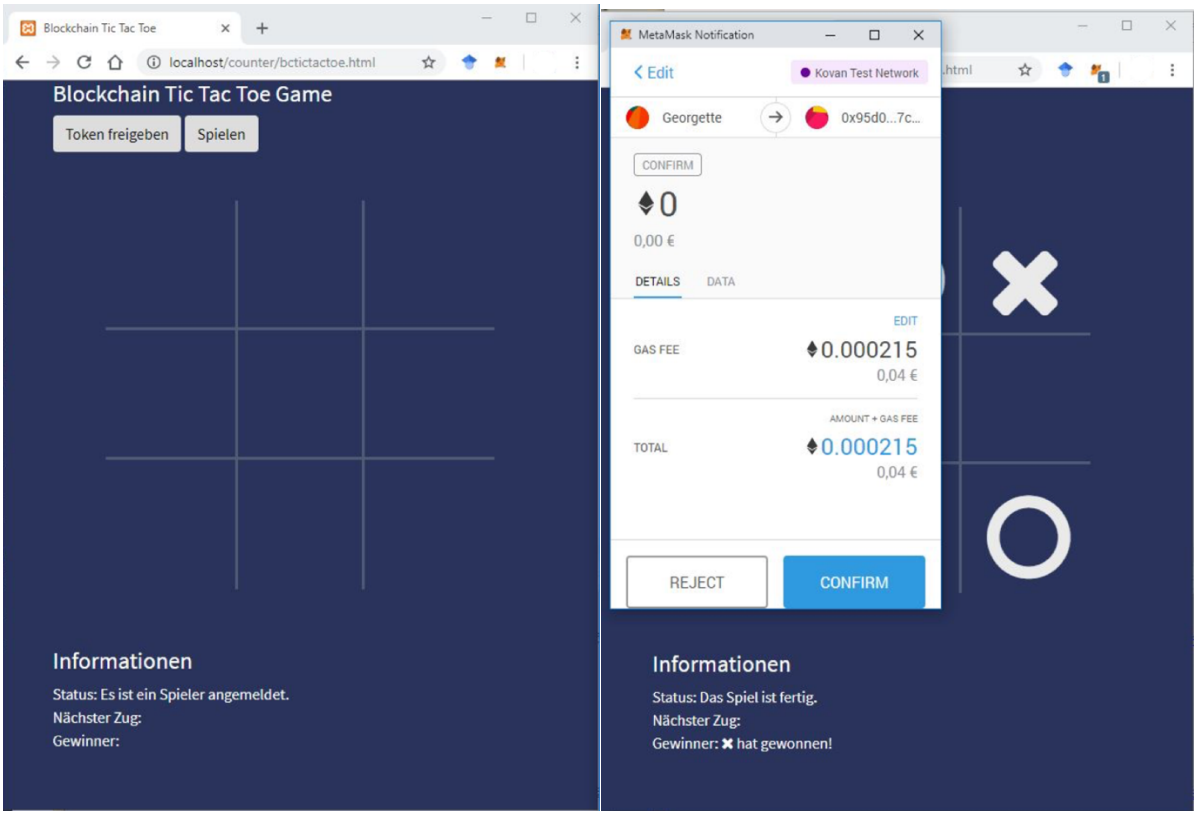

Abbildung 23: Webpage TicTacToe (links: Startbild, rechts: Spielzug)

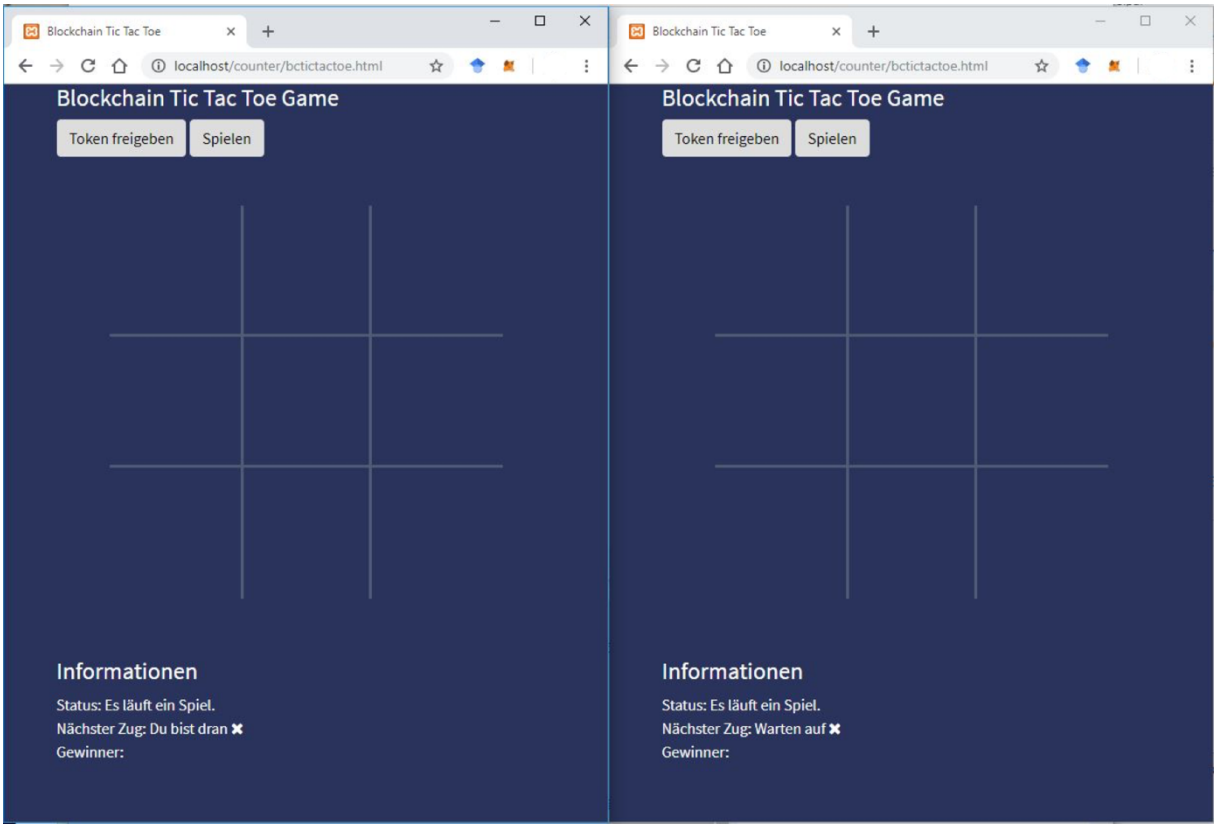

Abbildung 24: TicTacToe Spiel mit zwei Spielern (2 Browser Fenster)

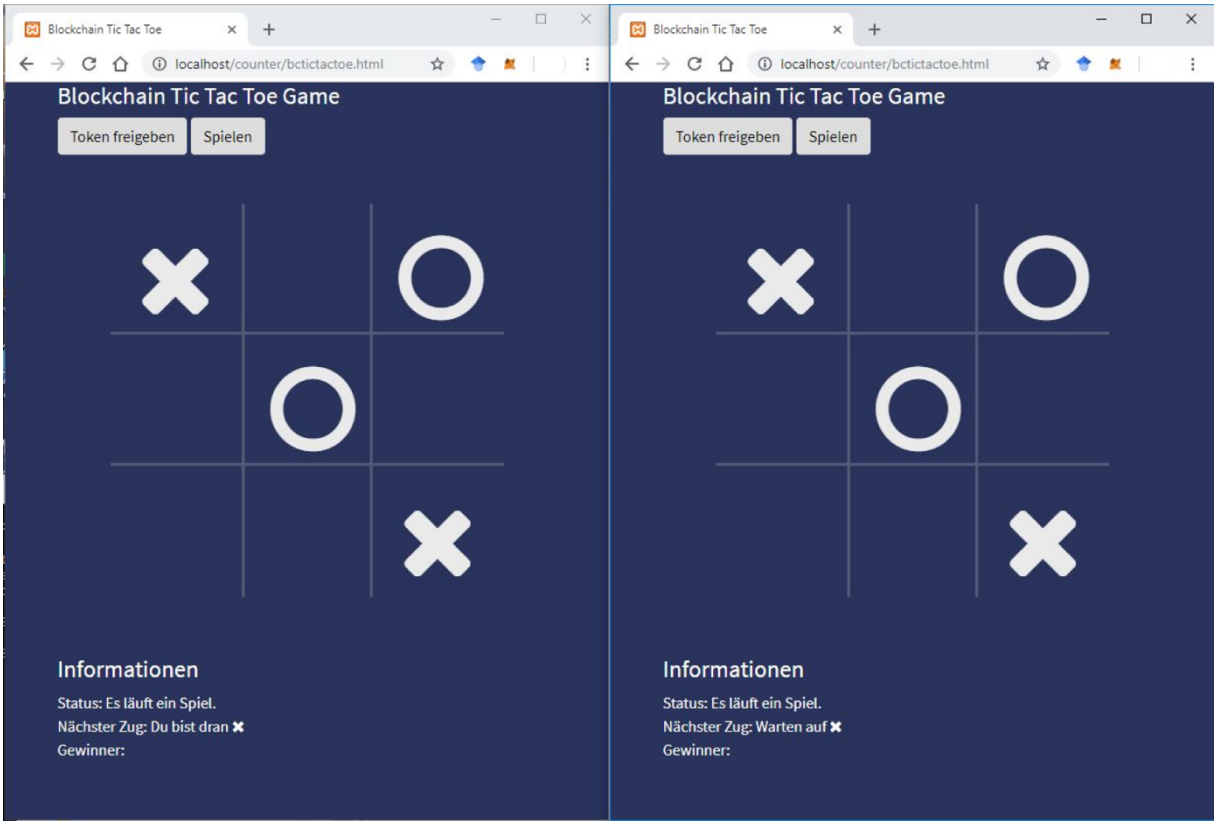

Abbildung 25: TicTacToe mit zwei Spielern (links: Spieler X, rechts: Spieler O)

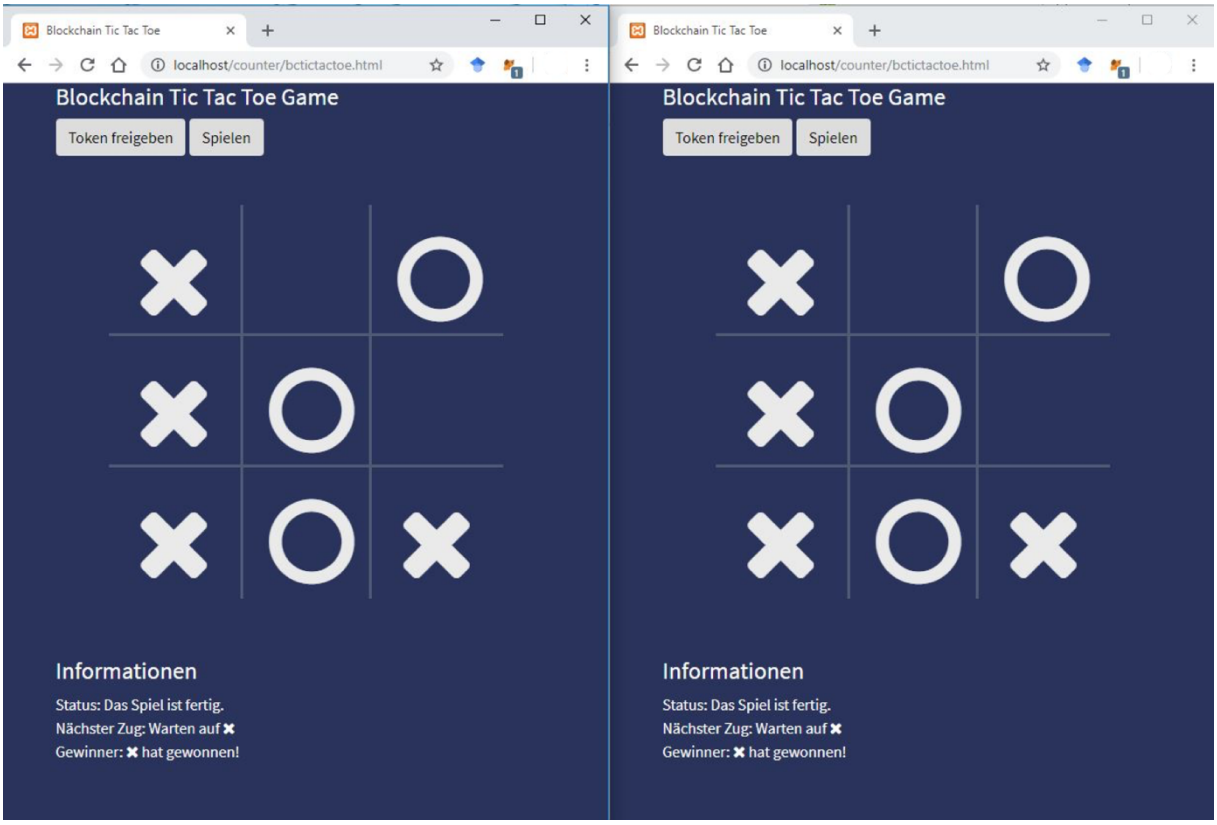

Abbildung 26: Spielende. X hat gewonnen.

4.5. Fragen

18. Was ist der Nutzen davon, Programmcode auf einer Blockchain abzuspeichern?
19. Für diese Aufgabe ist die eigene Kreativität gefragt. Welche Mittelsmänner könnten Smart Contracts in Zukunft alles ablösen? Wie sehen die groben Konzepte hinter diesen Smart Contracts aus?
20. Was ist der Unterschied zwischen ERC20 Tokens und ERC721 Tokens?
21. Anhand des Codes im TicTacToe Contract soll hier ein eigenes Programm entwickelt werden. Eine Zahl namens Counter soll durch eine Funktion um eins vergrössert werden. Eine zweite Funktion soll den Counter um eins verringern. Eine letzte Funktion soll den Counter als Ausgabewert haben.

5. SCHLUSSWORT

Wir haben in der Maturaarbeit ausführlich dargestellt, wie Mittelsmänner durch die Blockchain Technologie ausgeschaltet werden können. Dabei konnten wir verstehen, was die einzelnen Bestandteile der Blockchain sind, wie sie funktionieren und wofür die Blockchain gebraucht wird beziehungsweise in Zukunft gebraucht werden kann. Wir haben die unglaublich faszinierenden Verfahren des «Digital Signature Algorithms» und der «Hashfunktion» gesehen und verstanden. Weiter sahen wir, wie im Netzwerk Informationen verbreitet werden, wieso die Blockchain sicher ist und was «Smart Contracts» sind und wie sie funktionieren. Dabei haben wir die vielen Vorteile gesehen, aber auch die Nachteile und mögliche Probleme wie zum Beispiel das Double Spending Problem.

Die öffentliche Aufmerksamkeit fokussiert sich im Moment vor allem auf Kryptowährungen, wir sind jedoch sicher, dass vor allem die Wichtigkeit der Smart Contracts zunehmen wird, da diese stark anwendungsorientiert sind. Im zweiten Teil der Arbeit, im selbst programmierten TicTacToe Beispiel wurde ersichtlich, was eine solche Anwendung sein könnte. Hier konnten wir auch zeigen, wie die Programmierung und die Benutzerinteraktion funktionieren. Auf derartigen Anwendungen lässt sich aufbauen, um der Technologie immer komplexere Aufgaben zu übergeben.[61]

Wir sind überzeugt, dass die Blockchain an Anerkennung gewinnen wird. Vor allem überzeugen uns die faszinierenden Möglichkeiten, welche die Blockchain bietet, um ein Netzwerk ohne jegliches Vertrauen möglich macht. Dies bringt in Zukunft sicherlich sehr spannende, neue Möglichkeiten. Das Ziel dieser Arbeit ist es, diese Meinung weiterzuvermitteln und zu zeigen wie vielfältig die Anwendungen der Blockchain sind und in Zukunft sein werden. Auch hoffen wir, dass immer mehr Leute beim Namen „Blockchain“ nicht mehr ausschliesslich Bitcoin vor sich sehen, sondern ein breites Ausmass an spannenden Möglichkeiten und Anwendungen sehen und den Wert dieser Anwendungen verstehen. Der Erfinder von Ethereum, Vitalik Buterin, sagte bezüglich der Blockchain:

“Whereas most technologies tend to automate workers on the periphery doing menial tasks, blockchains automate away the center. Instead of putting the taxi driver out of a job, blockchain puts Uber out of a job and lets the taxi drivers work with the customer directly.”[62]
- Vitalik Buterin

Wir bedanken uns ganz herzlich bei Armin Barth, welcher uns als Erstbetreuer bei inhaltlichen Fragen stets und motivierend zur Seite gestanden ist und unsere Arbeit durch seine Erklärungen zu mathematischen Aspekten sehr bereichert hat. Auch möchten wir uns ausdrücklich bei Thibault Meunier bedanken, welcher uns beim Erlernen von Solidity unterstützt hat und unsere Fragen zu dem TicTacToe Programm beantwortet hat. Für die Vermittlung des Kontakts zu Thibault Meunier und dem Beantworten von Fragen bezüglich html-Webpages danken wir Tim Weingärtner.

[61] Talin: Blockchain, heruntergeladen am 11.10.2018.
[62] Buterin: Blockchain, heruntergeladen am 11.10.2018.

6. QUELLENVERZEICHNIS

[1] 1500 Entwicklung des Bankenwesens, in: MyBude, http://www.mybude.com/renaissance-reformation/1500-entwicklung-des-bankwesens.html, heruntergeladen am 10.6.2018.

[2] Keuper, Ralf: Die Anfänge des modernen Bankwesens in Oberitalien (Film), https://bankstil.de/die-anfange-des-modernen-bankwesens-in-oberitalien-film, heruntergeladen am 10.6.2018.

[3] Banken, in: Historisches Lexikon der Schweiz, http://www.hls-dhs-dss.ch/textes/d/D14061.php, heruntergeladen am 10.6.2018.

[4] Aufgaben der Bank, in: Jugend und Bildung, http://www.jugend-und-bildung.de/files/284/Aufgaben_einer_Bank_07_08.pdf, heruntergeladen am 10.6.2018.

[5] Gebührentarif, in: Alternative Bank Schweiz, https://www.abs.ch/?id=622, heruntergeladen am 10.6.2018.

[6] Build Decentralized Energy Communities Secured by the Blockchain, in: HivePower, https://www.hivepower.tech/, heruntergeladen am 10.6.2018.

[7] McGew, Matt: The Disadvantages of a Centralized Network Scheme,https://itstillworks.com/disadvantages-centralized-network-scheme-12213044.html, heruntergeladen am 22.8.2018.

[8] Prof. Dr. Bendel, Oliver: Kryptowährungen, Definition, https://wirtschaftslexikon.gabler.de/definition/kryptowaehrung-54160, heruntergeladen am 17.7.2018.

[9] Nakamoto, Satoshi: Bitcoin: A Peer-to-Peer Electronic Cash System, https://bitcoin.org/bitcoin.pdf, heruntergeladen am 30.5.2018.

[10] Die Geschichte des Bitcoins, in: moneymuseum, https://www.moneymuseum.com/pdf/PictureTours_bitcoin/Geschichte%20von%20Bitcoin_de-shrunk.pdf, heruntergeladen am 20.5.2018.

[11] Bitcoin – Schweizer Franken, in: finanzen.ch, https://www.finanzen.ch/devisen/bitcoin-franken-kurs, heruntergeladen am 19.9.2018.

[12] Bower, Bruce: How an ancient stone money system works like cryptocurrency. https://www.sciencenews.org/article/yap-stone-money-bitcoin-blockchain-cryptocurrency, heruntergeladen am 26.10.18.

[13] Keuper, Ralph: Das Steingeld von Mikronesien. Vorläufer der Blockchain, https://bankstil.de/das-steingeld-von-mikronesien-yap-vorlaufer-der-blockchain, heruntergeladen am 26.10.18.

[14] Nenavath Santhosh: Blockchain Demystified YAP's intuition. 4.04.2017. https://www.youtube.com/watch?v=Fy8BfVrj4dk, heruntergeladen am 26.10.18.

[15] Peer to Peer Network, in: Lisk, https://lisk.io/academy/blockchain-basics/how-does-blockchain-work/what-is-a-peer-to-peer-network, heruntergeladen am 22.08.18.

[16] Drescher, Daniel: Blockchain Basics. A Non-Technical Introduction in 25 Steps, Frankfurt am Main: Apress, Jahr 2017.

[17] Secure Hash Algorithm (SHA), in: Techopedia, https://www.techopedia.com/definition/10328/secure-hash-algorithm-sha, heruntergeladen am 12.10.2018.

[18] Drescher, Daniel: Blockchain Basics. A Non-Technical Introduction in 25 Steps, Frankfurt am Main: Apress, Jahr 2017.

[19] lerntool.ch: Hexadezimale Addition. 09.03.2015. https://www.youtube.com/watch?v=1rtCJgNlE-8, heruntergeladen am 12.10.2018.

[20] The Tech Train: Understanding ASCII and Unicode. 07.12.2017. https://www.youtube.com/watch?v=5aJKKgSEUnY, heruntergeladen am 25.10.2018.

[21] The cryptographic hashfunction SHA-256, in: researchgate, https://www.researchgate.net/file.PostFileLoader.html?id=534b393ad3df3e04508b45ad&assetKey=AS%3A273514844622849%401442222429260, heruntergeladen am 22.08.2018.

[22] Das Geburtstagsphänomen, in: staff.uni-mainz, https://www.staff.uni-mainz.de/pommeren/Kryptologie/Klassisch/2_Polyalph/GebPhaen.pdf, heruntergeladen am: 23.10.2018.

[23| Pommerening Klaus/Sergl Marita: Hash-Funktionen (kryptographische Prüfsummen, Message Digest). https://www.staff.uni-mainz.de/pommeren/DSVorlesung/KryptoBasis/Hash.html, heruntergeladen am: 23.10.2018.

[24] Pommerening Klaus: Die Sicherheit kryptographischer Verfahren. https://www.staff.uni-mainz.de/pommeren/DSVorlesung/KryptoBasis/KryptSicherheit.html, heruntergeladen am: 23.10.2018.

[25] Miessler, Daniel: Identification, Authentification and Authorisation, https://danielmiessler.com/blog/security-identification-authentication-and-authorization/, heruntergeladen am 16.03.18.

[26] Drescher, Daniel: Blockchain Basics. A Non-Technical Introduction in 25 Steps, Frankfurt am Main: Apress, Jahr 2017.

[27] Die Digitale Signatur, in: tu-chemnitz, https://www.tu-chemnitz.de/informatik/ThIS/downloads/courses/ss02/ebanking/schwartz2.pdf, heruntergeladen am 28.03.18.

[28] CuriousInventor: How Bitcoin works under the hood. 14.07.13. https://www.youtube.com/watch?v=Lx9zgZCMqXE, heruntergeladen am 28.03.18.

[29] Rehn, Christian: Restklassenringe. In: Software and Poetry, veröffentlicht am: 17.01.2012, heruntergeladen am 07.03.2018.

[30] Prof. Dr. Gubler, Walter: Lineare Algebra. Vorlesungsskript, http://www.mathematik.uni-regensburg.de/gubler/LineareAlgebra.pdf, heruntergeladen am 18.03.2018.

[31] Ganter Bernhard, Rechnen Modulo n. http://www.math.tu-dresden.de/~ganter/inf0708/folien/inf07-Modulo.pdf, heruntergeladen am 08.10.18.

[32] Buchmann, Johannes: The Digital Signature Algorithm. https://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1003_DSA.pdf, heruntergeladen am 16.03.2018.

[33] CuriousInventor: How Bitcoin works under the hood. 14.07.13. https://www.youtube.com/watch?v=Lx9zgZCMqXE, heruntergeladen am 28.03.18.

[34] Simply Explained – Savjee: How does a blockchain work – Simply Explained. 13.11.17. https://www.youtube.com/watch?v=SSo_EIwHSd4, heruntergeladen am 28.03.18.

[35] Brownworth, Anders: Blockchain Demo. https://anders.com/blockchain/blockchain.html, heruntergeladen am 30.05.18.

[36] Drescher, Daniel: Blockchain Basics. A Non-Technical Introduction in 25 Steps, Frankfurt am Main: Apress, Jahr 2017.

[37] Gossip data dissemination protocol, in: Hyperledger Fabric, https://hyperledger-fabric.readthedocs.io/en/release-1.3/gossip.html, heruntergeladen am 22.10.2018

[38] Demers, Alan/Gealy, Mark/Greene, Dan/Hauser, Carl/Irish, Wes/Larson, John/Manning, Sue/Shenker, Scott/ Sturgis, Howard/Swinehart, Dan/ Terry, Doug/ Woods, Don: Epidemic Algorithms for Replicated Database Maintenance. Palo Alto, California: Xerox Corporation, 1989 (Zweite Auflage).

[39] Drescher, Daniel: Blockchain Basics. A Non-Technical Introduction in 25 Steps, Frankfurt am Main: Apress, 2017.

[40] Tanenbaum Andrew S./Van Steen, Maarten: Distributed Systems. Principles and Paradigms, Upper Saddle River, NJ: Pearson, 2007 (Zweite Auflage).

[41] The Morpheus Tutorials: Kryptographie #63 – Merkle Trees. 16.09.2016. https://www.youtube.com/watch?v=yX2VoXHyTgI, heruntergeladen am 26.09.2018.

[42] Curran, Brian: What is a Merkle Tree?. Beginner's Guide to this Blockchain Component, https://blockonomi.com/merkle-tree/, heruntergeladen am 26.09.2018.

[43] Schiller: Merkle Tree. Eine Basis der Blockchain, https://blockchainwelt.de/merkle-tree-basis-von-blockchain-und-hash-trees/, heruntergeladen am 26.09.2018.

[44] Ray, Shaan: Merkle Trees. https://hackernoon.com/merkle-trees-181cb4bc30b4, heruntergeladen am 14.10.2018.

[45] Drescher, Daniel: Blockchain Basics. A Non-Technical Introduction in 25 Steps, Frankfurt am Main: Apress, Jahr 2017.

[46] Norma Uriarte: Double Spending Bitcoins. 08.09.2014. https://www.youtube.com/watch?v=3iYzpdU67D4, heruntergeladen am 09.10.2018.

[47] 51% Attack, in: Investopedia, https://www.investopedia.com/terms/1/51-attack.asp, heruntergeladen am 09.10.2018.

[48] Asolo, Bisade: 51% Attack Explained. 29.07.2018. https://www.mycryptopedia.com/51-percent-attack-explained/, heruntergeladen am 17.10.2018.

[49] Definition Vertrag, in: business-on, http://www.business-on.de/definition-vertrag-vereinbarung-zwischen-zwei-parteien-_id50657.html, heruntergeladen am 03.10.2018.

[50] Airbnb, in: Wikipedia, https://de.wikipedia.org/wiki/Airbnb#Konzept, heruntergeladen am 03.10.2018.

[51] Build Decentralized Energy Communities Secured by the Blockchain, in: hivepower, https://www.hivepower.tech/, heruntergeladen am 26.10.2018.

[52] Industry Focused + Cutting-Edge, in: modum, https://modum.io/solution/services, heruntergeladen am 26.10.2018.

[53] Blockchain Basics – What are Smart Contracts, in: medium, https://medium.com/coinmonks/blockchain-basics-what-is-a-smart-contract-ddd99c50b850, heruntergeladen am 03.10.2018.

[54] Hoffmann, Chris: What is Ethereum, and What are Smart Contracts, https://www.howtogeek.com/350322/what-is-ethereum-and-what-are-smart-contracts/, heruntergeladen am 03.10.2018.

[55] Simply Explained – Savjee: Smart contracts – Simply Explained, 20.11.17. https://www.youtube.com/watch?v=ZE2HxTmxfrI, heruntergeladen am 03.10.2018.

[56] What is a Cryptocurrency Token?, in: CryptoCurrency Facts, https://cryptocurrencyfacts.com/what-is-a-cryptocurrency-token/, heruntergeladen am 11.10.2018.

[57] The Blockchain Bubble's Latest Victim: Digital Cats, in: Fortune, http://fortune.com/2018/06/18/cryptokitties/, heruntergeladen am 11.10.2018.

[58] Create your own crypto-currency with Ethereum, in: Ethereum, https://ethereum.org/token, heruntergeladen am 12.07.2018

[59] Kaluzny: Tic Tac Toe, https://github.com/mrkaluzny/tic-tac-toe, https://mrkaluzny.com/tic-tac-toe-javascript-game/, heruntergeladen am 29.09.2018

[60] Ethereum Contract ABI, in: github, https://github.com/ethereum/wiki/wiki/Ethereum-Contract-ABI, heruntergeladen am 26.10.2018.

[61] Talin, Benjamin: Blockchain. Möglichkeiten Und Anwendungen Der Technologie, https://morethandigital.info/blockchain-moeglichkeiten-und-anwendungen-der-technologie/, heruntergeladen am 11.10.2018.

[62] Butarin, Vitalik: Blockchain. https://www.brainyquote.com/topics/blockchain, heruntergeladen am 11.10.2018.

7. ABBILDUNGSVERZEICHNIS

8. STICHWORTVERZEICHNIS

9. ANHANG: CODE DES SMART CONTRACTS

```
pragma solidity ^0.4.24;

contract ERC20Interface {
   function totalSupply() public constant returns (uint256 );
   function balanceOf(address _owner) public constant returns (uint256);
   function transfer(address _to, uint256 _value) public returns (bool success);
   function transferFrom(address _from, address _to, uint256 _value) public re-
turns (bool success);
   function approve(address _spender, uint256 _value) public returns (bool suc-
cess);
   function allowance(address _owner, address _spender) public constant returns
(uint256 remaining);
   event Transfer(address indexed _from, address indexed _to, uint256 _value);
   event Approval(address indexed _owner, address indexed _spender, uint256
_value);
}

contract Fields {
   //if someone wants to choose a field, we need to make sure that it isn't occu-
pied yet
   // i describes the row, k the column

   uint8 [3][3] internal fields;
   bool internal endGame = true; // true means game has already ended (default
Value is true)
   string internal winnerOfGame = "";

   function fieldChange (uint8 i, uint8 k, uint8 value) internal { //not sure if I
used enum correctly here
       fields[i-1][k-1] = value;
   } //this function is called when sb. wants to choose a field

   function getFieldValue(uint8 i, uint8 k) view public returns (uint8) {
       return fields[i-1][k-1];
   }

   function getAllFieldValues() view public returns (uint256) {
       uint8 i;
       uint8 k;
       uint256 result=0;
       uint256 position=0;
       for (i=0; i<=2; i++) {
           for (k=0; k<=2; k++) {
               result = result + (fields[i][k] * (10**position));
               position++;
           }
       }
       return result;
   }
}
```

```
contract User is Fields {
   ERC20Interface internal KantiToken;
   address internal PlayerA;
   address internal PlayerB;
   mapping (address => uint8) internal playerSymbol;

   uint8 internal playerDefined = 0;

   function DefineUsers() public {
       require(endGame == true);
       require(PlayerB == 0x0000000000000000000000000000000000000000);

       if (PlayerA == 0x0000000000000000000000000000000000000000) {
           PlayerA = msg.sender;
           playerSymbol[PlayerA] = 1;
           playerDefined = 1;
                   // check if tranfer allowed by sender
                   if(!(KantiToken.allowance(msg.sender,   this)   >   0))   re-
vert('Transfer not allowed!');
           //check if the transaction was received
                   if(!KantiToken.transferFrom(msg.sender, this, uint256(1))) re-
vert('Transfer aborded!');//KantiToken.transfer(this, 100000000000000000);
           // init fields
           for (uint8 i=0; i<=2; i++) {
               for (uint8 k=0; k<=2; k++) {
                   fields[i][k] = 0;
               }
           }
       }
       else if (PlayerA != msg.sender) {
           PlayerB = msg.sender;
           playerSymbol[PlayerB] = 2;
           playerDefined = 2;
           endGame = false;
                   // check if tranfer allowed by sender
                   if(!(KantiToken.allowance(msg.sender,   this)   >   0))   re-
vert('Transfer not allowed!');
                   //check if the transaction was received
           if(!KantiToken.transferFrom(msg.sender,    this,    uint256(1)))    re-
vert('Transfer aborded!');//KantiToken.transfer(this, 100000000000000000);
       }
             // revert will abord the function
   }

//needs a lock, so functions can't be called while game is still going on

   function getPlayerA() view public returns (address) {
       return PlayerA;
   }

   function getPlayerB() view public returns (address) {
       return PlayerB;
   }
}

/*   k=1 k=2 k=3
    _____________  Plan
i=1 |___|___|___|
i=2 |___|___|___|
i=3 |___|___|___|
*/
```

```
contract Winner is User {
   uint count = 0;
   uint8 internal helpWinner = 0; //provisorisch recognizing Winner

   function wonGameColumnk (uint playerValue, uint column) internal returns (bool)
{
//column k
       count = 0;
       for (uint8 i=1; i<=3; i++) {
           if (fields[i-1][column-1] == playerValue) {
               count++;
           }
       }

       if (count == 3) {
           return true;
       }
       else {
           return false;
       }
   }

   function wonGameRowi (uint playerValue, uint row) internal returns (bool) {
//row i
       count = 0;
       for (uint8 k=1; k<=3; k++) {
           if (fields[row-1][k-1] == playerValue) {
               count++;
           }
       }
       if (count == 3) {
           return true;
       }
       else {
           return false;
       }
   }

   function wonGameDiagonal (uint8 playerValue) internal returns (bool) {
//bottom left to top right
       count = 0;
       uint8 i = 3;
       for (uint8 k=0; k<=2; k++) {
           i = i - 1;
           if (fields[i][k] == playerValue) {
               count++;
           }
       }
       if (count == 3) {
           return true;
       }

//top left to bottom right
       count = 0;
       i = 0;
       for ( k=0; k<=2; k++) {
           if (fields[i][k] == playerValue) {
               count++;
           }
           i++;
```

```
        }

        if (count == 3) {
            return true;
        }
        else {
            return false;
        }
    }
}

contract playGame is Winner {
    uint8 internal whichPlayer = 1;
    uint8 internal moveCount = 0;
    uint256 playTime = 0;

    event gameOver(
        address Winner,
        string message
    );

    event Error(string mistake); // is emitted if sb. won the gameOver

    function checkTime() public {
        if (!endGame && now > playTime + 10 minutes) {
            if (whichPlayer == PlayerA) {
                winner(PlayerB);
            }
            if (whichPlayer == PlayerB) {
                winner(PlayerA);
            }
        }
    }

       constructor(address _KantiToken) public {
                //ERC20Interface                  ERC20Contract                  =
ERC20Interface(0x71ac80ee659737dc5feeebb2c935d009a35b1849);
              KantiToken = ERC20Interface(_KantiToken);
    }


    function statusInit()  view public returns (uint8) {
        return playerDefined;

    }

    function whoseTurn() view public returns (uint8) {
        if (whichPlayer == 1) {
            return 1;
        }
        else {
            return 2;
        }
    }

    function checkWinner () view public returns (uint8) {
        if (endGame) {
            return helpWinner;
        }
        else {
            return 0;
```

```
    }
}

    function whoAmI() view public returns (uint8) {
    if (PlayerA == msg.sender) {
        return 1;
    }
    else if (PlayerB == msg.sender) {
        return 2;
    }
    else{
        return 0;
    }
}

function getTime() view public returns (uint256) {
    return playTime;
}


function Play(uint8 i, uint8 k) public {
    require(playerDefined == 2);
    uint8 playerValue = playerSymbol[msg.sender];

    if (whichPlayer == playerValue) {
        if (fields[i-1][k-1] == 0) {
            fieldChange (i, k, playerSymbol[msg.sender]);
            if (msg.sender == PlayerA) {
                whichPlayer = 2;
            }
            else {
                whichPlayer = 1;
            }
            moveCount ++;
            playTime = now;
            //PlayerB is allowed to play now
            //checking if PlayerA has won:
            if (wonGameColumnk(playerValue, k) == true) {
                winner(msg.sender);
            }
            if (wonGameRowi(playerValue, i) == true) {
                winner(msg.sender);
            }
            if (wonGameDiagonal(playerValue) == true) {
                winner(msg.sender);
            }
            if (moveCount == 9) {
                winnerNobody();
            }
        }
        else {
            emit Error("field is already occupied");
        }
    }

    else {
        emit Error("it's not your turn yet");
    }

} //function which is called by PlayerA to choose field i,k
```

```
    function reset() internal {
        endGame = true;
        playerDefined = 0;
        PlayerA = 0x0000000000000000000000000000000000000000;
        PlayerB = 0x0000000000000000000000000000000000000000;
        whichPlayer = 1;
        moveCount = 0;
    }

    function winner(address checkWinnerId) internal {
        emit gameOver(checkWinnerId, "has won the Game");
              if(!KantiToken.transfer(checkWinnerId, uint256(2))) revert();

        if (checkWinnerId == PlayerA) {
            helpWinner = 1;
        }

        else {
            helpWinner = 2;
        }
        reset();
    }

    function winnerNobody() internal {
        helpWinner = 3;
              if(!KantiToken.transfer(PlayerA, uint256(1))) revert();
              if(!KantiToken.transfer(PlayerB, uint256(1))) revert();
        reset();
    }
}
```

10. ANHANG: CODE DES KANTI TOKEN

```
pragma solidity ^0.4.16;

interface tokenRecipient { function receiveApproval(address _from, uint256 _value,
address _token, bytes _extraData) external; }

contract TokenERC20 {
    // Public variables of the token
    string public name;
    string public symbol;
    uint8 public decimals = 18;
    // 18 decimals is the strongly suggested default, avoid changing it
    uint256 public totalSupply;

    // This creates an array with all balances
    mapping (address => uint256) public balanceOf;
    mapping (address => mapping (address => uint256)) public allowance;

    // This generates a public event on the blockchain that will notify clients
    event Transfer(address indexed from, address indexed to, uint256 value);

    // This generates a public event on the blockchain that will notify clients
    event  Approval(address  indexed  _owner,  address  indexed  _spender,  uint256
_value);

    // This notifies clients about the amount burnt
    event Burn(address indexed from, uint256 value);

    /**
     * Constructor function
     *
     * Initializes contract with initial supply tokens to the creator of the con-
tract
     */
    function TokenERC20(
        uint256 initialSupply,
        string tokenName,
        string tokenSymbol
    ) public {
        totalSupply = initialSupply * 10 ** uint256(decimals);   // Update total
supply with the decimal amount
        balanceOf[msg.sender] = totalSupply;                    // Give the creator
all initial tokens
        name = tokenName;                                       // Set the name for
display purposes
        symbol = tokenSymbol;                                  // Set the symbol for
display purposes
    }

    /**
     * Internal transfer, only can be called by this contract
     */
    function _transfer(address _from, address _to, uint _value) internal {
        // Prevent transfer to 0x0 address. Use burn() instead
        require(_to != 0x0);
        // Check if the sender has enough
```

```
        require(balanceOf[_from] >= _value);
        // Check for overflows
        require(balanceOf[_to] + _value >= balanceOf[_to]);
        // Save this for an assertion in the future
        uint previousBalances = balanceOf[_from] + balanceOf[_to];
        // Subtract from the sender
        balanceOf[_from] -= _value;
        // Add the same to the recipient
        balanceOf[_to] += _value;
        emit Transfer(_from, _to, _value);
        // Asserts are used to use static analysis to find bugs in your code. They
should never fail
        assert(balanceOf[_from] + balanceOf[_to] == previousBalances);
    }

    /**
     * Transfer tokens
     *
     * Send `_value` tokens to `_to` from your account
     *
     * @param _to The address of the recipient
     * @param _value the amount to send
     */
    function transfer(address _to, uint256 _value) public returns (bool success) {
        _transfer(msg.sender, _to, _value);
        return true;
    }

    /**
     * Transfer tokens from other address
     *
     * Send `_value` tokens to `_to` on behalf of `_from`
     *
     * @param _from The address of the sender
     * @param _to The address of the recipient
     * @param _value the amount to send
     */
    function transferFrom(address _from, address _to, uint256 _value) public re-
turns (bool success) {
        require(_value <= allowance[_from][msg.sender]);     // Check allowance
        allowance[_from][msg.sender] -= _value;
        _transfer(_from, _to, _value);
        return true;
    }

    /**
     * Set allowance for other address
     *
     * Allows `_spender` to spend no more than `_value` tokens on your behalf
     *
     * @param _spender The address authorized to spend
     * @param _value the max amount they can spend
     */
    function approve(address _spender, uint256 _value) public
        returns (bool success) {
        allowance[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);
        return true;
    }

    /**
```

```
     * Set allowance for other address and notify
     *
     * Allows `_spender` to spend no more than `_value` tokens on your behalf, and
then ping the contract about it
     *
     * @param _spender The address authorized to spend
     * @param _value the max amount they can spend
     * @param _extraData some extra information to send to the approved contract
     */
    function approveAndCall(address _spender, uint256 _value, bytes _extraData)
        public
        returns (bool success) {
        tokenRecipient spender = tokenRecipient(_spender);
        if (approve(_spender, _value)) {
            spender.receiveApproval(msg.sender, _value, this, _extraData);
            return true;
        }
    }

    /**
     * Destroy tokens
     *
     * Remove `_value` tokens from the system irreversibly
     *
     * @param _value the amount of money to burn
     */
    function burn(uint256 _value) public returns (bool success) {
        require(balanceOf[msg.sender] >= _value);    // Check if the sender has
enough
        balanceOf[msg.sender] -= _value;            // Subtract from the sender
        totalSupply -= _value;                      // Updates totalSupply
        emit Burn(msg.sender, _value);
        return true;
    }

    /**
     * Destroy tokens from other account
     *
     * Remove `_value` tokens from the system irreversibly on behalf of `_from`.
     *
     * @param _from the address of the sender
     * @param _value the amount of money to burn
     */
    function burnFrom(address _from, uint256 _value) public returns (bool success)
{
        require(balanceOf[_from] >= _value);                // Check if the tar-
geted balance is enough
        require(_value <= allowance[_from][msg.sender]);    // Check allowance
        balanceOf[_from] -= _value;                         // Subtract from the
targeted balance
        allowance[_from][msg.sender] -= _value;             // Subtract from the
sender's allowance
        totalSupply -= _value;                              // Update totalSupply
        emit Burn(_from, _value);
        return true;
    }
}
```

11. ANHANG: CODE DER WEBPAGE

```
<!DOCTYPE html>
<html>
    <head>
        <title>Blockchain Tic Tac Toe</title>
        <link href="css/bootstrap.min.css" rel="stylesheet">
        <link href="css/font-awesome.css" rel="stylesheet">
        <link href="css/style.css" rel="stylesheet">
        <!-- Script -->
        <script src="js/jquery.min.js"></script>
        <script src="js/bootstrap.min.js"></script>

       <script>

       var GameAddress = '0x4a08c7c24ba7ed928387fe4dc76b292a17b7ba51';

       var KantiTokenAddress = '0x89d4fe5359a4fd275b356732ed8d81558a228048';
       if (typeof window.web3 !== 'undefined') {
              // Use Mist/MetaMask's provider
              var web3js = new Web3(web3.currentProvider);
       }
       else {
              alert("No MetaMask defined!");
       }

       var playGameContract = web3js.eth.contract(
//// ABI des Game Contracts
       );
       var kantiToken = web3js.eth.contract(
//// ABI des Kanti Token Contracts
       );

       // Address of deployed contract
       var playGameContractInstance = playGameContract.at(GameAddress);
       var kantiTokenInstance = kantiToken.at(KantiTokenAddress);
       var playerId = 0;        // save player id
       var winShowed = false;

       // Fill the game field
       function fillFields() {
       /*
         [1, 2, 3],
         [4, 5, 6],
         [7, 8, 9]
                */
       // add leading 0 from the smart contract function
              var s = "";
              playGameContractInstance.getAllFieldValues(
                     function(err, res){
                            if (!err) {
                                   s = res.c[0]+"";
                                   while (s.length < 9) s = "0" + s;
                                   console.log('all Fields: ',s);
                                   var helper=0;
                                   for (i=1; i<=9; i++) {
                                          helper = parseInt(s.substr(9-i,1));
                                          if (helper == 1){
                                                 $('#' + i).html('<span class="fa fa-
times"></span>');
                                          }
                                          else if (helper == 2) {
                                                 $('#' + i).html('<span class="fa fa-
circle-o"></span>');
                                          }
                                          else {
```

```
                                        $('#' + i).html('');
                                }
                        }
                }
                else
                        console.log(err);
        });

        // check if somebody won
        var win =0;
        playGameContractInstance.checkWinner(
                function(err, res){
                        if (!err) {
                                console.log('winner: ',res.c[0]);
                                win = res.c[0];
                                if (!winShowed && win == 1) {
                                        $('#showWin').html('<span class="fa fa fa-
times"></span> hat gewonnen!');
                                        winShowed = true;
                                }
                                else if (!winShowed && win == 2) {
                                        $('#showWin').html('<span class="fa fa-circle-
o"></span> hat gewonnen!');
                                        winShowed = true;
                                }
                                else if (!winShowed && win == 3) {
                                        $('#showWin').html('Kein gewinner!');
                                        winShowed = true;
                                }
                        }
                        else
                                console.log(err);
        });

        // check who has the next draw
        var status = 0;
        playGameContractInstance.statusInit(
                function(err, res){
                        if (!err) {
                                console.log('statusInit: ',res.c[0]);
                                status = parseInt(res.c[0]);
                                if (status == 0) {
                                        $('#showState').html('Das Spiel ist fertig.');
                                        winShowed = false;
                                }
                                else if (status == 1) {
                                        $('#showState').html('Es ist ein Spieler ange-
meldet.');
                                        winShowed = true;
                                        $('#showWin').html('');
                                }
                                else if (status > 1) {
                                        $('#showState').html('Es l&auml;uft ein
Spiel.');
                                        winShowed = true;
                                        $('#showWin').html('');
                                        var turn = 0;
        playGameContractInstance.whoseTurn(
                function(err, res){
                        if (!err) {
                                console.log('whose Turn: ',res.c[0]);
                                turn = parseInt(res.c[0]);

                                if (turn == 1 && playerId == 1) {
                                        $('#showTurn').html('Du bist dran <span class="fa fa-
times"></span>');
                                }
```

```
                        else if (turn == 2 && playerId == 2) {
                                $('#showTurn').html('Du bist dran <span class="fa fa-
circle-o"></span>');
                        }
                        else if (turn == 1 && playerId == 2) {
                                $('#showTurn').html('Warten auf <span class="fa fa-
times"></span> ');
                        }
                        else if (turn == 2 && playerId == 1) {
                                $('#showTurn').html('Warten auf  <span class="fa fa-
circle-o"></span> ');
                        }
                        else if (playerId == 0) {
                                if (turn == 1) {
                                        $('#showTurn').html('Warten auf <span class="fa
fa-times"></span> ');
                                }
                                else if (turn == 2) {
                                        $('#showTurn').html('Warten auf  <span class="fa
fa-circle-o"></span> ');
                                }
                        }
                    }
                    else
                        console.log(err);
        });
                                }
                        }
                        else
                                console.log(err);
                });
        }

        function approve() {
        // Allow transaction
                kantiTokenInstance.approve(GameAddress,
                parseInt(web3.toWei(1,'wei')),
                function(err, res){
                if (!err) {
                        console.log('result approve: ',res);
                }
                else
                        console.log(err);
                });
        }

        // initially set players
        function setPlayer() {
                var status = 0;

                // check status
                playGameContractInstance.statusInit(
                        function(err, res){
                                if (!err) {
                                        console.log('status Init set player: ',res.c[0]);
                                        status = parseInt(res.c[0]);
                                        if (status == 0) {
                                                playerId = 1;
                                        }
                                        else if (status == 1) {
                                                playerId = 2;
                                        }
                                        else if (status == 2) {
                                                playerId = 0;
                                                return;
                                        }
                                        // Set Player
```

```
                                        playGameContractInstance.DefineUsers(
                                                function(err,res) {
                                                if (!err) {
                                                        console.log('result define user =>
',res);
                                                }
                                                else
                                                        console.log(err);
                                        });
                                }
                                else {
                                        console.log(err);
                                }
                        });
        }

        // if icon is choosen
        function icon(id) {
                console.log("Field choosen ",id);
                switch (id) {
                        case '1':
                                playGameContractInstance.Play(1,1, function(err, res){
                                if (!err) {
                                        console.log('play11: ',res);
                                }
                                else {
                                        console.log(err);
                                }});
                                break;
                        case '2':
                                playGameContractInstance.Play(1,2, function(err, res){
                                if (!err) {
                                        console.log('play12: ',res);
                                }
                                else {
                                        console.log(err);
                                }});
                                break;
////// analog für die Felder 3 bis 9
                }
        }

        // update fields, status every 10 seconds
        function update() {
                fillFields();
                setTimeout(update,10000);
        }

        update();

        </script>
</head>
<body>
        <div class="container">
                <h4 style="color:white;"> Blockchain Tic Tac Toe Game </h4>
                <button type="button" class="btn" onClick="approve();">
                        Token freigeben
                </button>
                <button type="button" class="btn" onClick="setPlayer();">
                        Spielen
                </button>

                <div class="game">
                <div id="1" onClick="icon(this.id)"
                        class="game-field" style="border-bottom: 3px solid #525B76;"></div>
                <div id="2" onClick="icon(this.id)"
```

```
                class="game-field" style="border-bottom: 3px solid #525B76;border-
            left: 3px solid #525B76; border-right: 3px solid #525B76;"></div>
            <div id="3" onClick="icon(this.id)"
                class="game-field" style="border-bottom: 3px solid #525B76;"></div>
            <div id="4" onClick="icon(this.id)"
                class="game-field" style="border-bottom: 3px solid #525B76;"></div>
            <div id="5" onClick="icon(this.id)"
                class="game-field" style="border-bottom: 3px solid #525B76; border-
                left: 3px solid #525B76; border-right: 3px solid #525B76;"></div>
            <div id="6" onClick="icon(this.id)"
                class="game-field" style="border-bottom: 3px solid #525B76;"></div>
            <div id="7" onClick="icon(this.id)"
                class="game-field"></div>
            <div id="8" onClick="icon(this.id)"
                class="game-field" style="border-left: 3px solid #525B76; border-
            right: 3px solid #525B76;"></div>
            <div id="9" onClick="icon(this.id)"
                class="game-field"></div>
            </div>

            <div class="signature" style="color:white;">
                <h4>Informationen</h4>
                Status: <span id="showState" style="color:white;"> </span> <br>
                N&auml;chster Zug: <span id="showTurn" style="color:white;">
                </span> <br>
                Gewinner: <span id="showWin" style="color:white;"> </span><br>
            </div>
        </div>
</body>
</html>
```

12. ANHANG: ANTWORTEN

Kapitel 1: Einleitung

1. Blockchain ist die Technologie, welche es möglich macht Mittelsmänner zu umgehen. Bitcoin hingegen ist eine Anwendung der Blockchain. Das Konzept der Blockchain bestand bereits vor dem Bitcoin, Bitcoin läuft jedoch auf der ersten funktionierenden Blockchain.
2. Dadurch, dass kein Yapi die Aufgabe hat, alle Transaktionen zu überwachen, müssen keine Transaktionsgebühren verlangt werden. Auch ist es so nicht möglich, dass ein Yapi alleine die Macht über die Besitztümer der anderen hat. In diesem Fall wäre es nämlich fatal, wenn dem Yapi mit aller Macht etwas zustossen würde, er etwas vergessen würde oder er die anderen betrügen wollte. Damit das Konzept funktioniert, müssen die Yapis einander jedoch vertrauen und müssen ständig davon ausgehen, dass die anderen Yapis sie im Streitfall unterstützen. Dies wiederum ist nicht selbstverständlich.
3. Im Peer-to-Peer Netzwerk stellen alle Computer einen Teil ihrer Ressourcen zur Verfügung. Ein Beispiel für eine solche Ressource wäre Rechenleistung. Dadurch, dass sehr viele Computer im Peer-to-Peer Netzwerk ihre Rechenleistung zur Verfügung stellen, können Berechnungen schneller durchgeführt werden, als ein einzelner Computer dies schaffen würde.

Kapitel 2: Grundlagen

4. a) 01000010 01101100 01101111 01100011 01101011 01100011 01101000 01100001 01101001 01101110

   b) W_1 = 01000010011011000110111101100011
   W_2 = 01101011011000110110100001100001
   W_3 = 01101001011011110000000000000000
   .
   .
   .
   W_{16} = 00000000000000000000000001010000

   c) W_{17} = 01011000000011110110000001100111

   d) H_1 = 6a09667

5. Es bräuchte $\sqrt{16^6} = 4096$ Versuche.
6. Mit dem privaten Schlüssel kann man beweisen, dass man der Besitzer einess Accounts ist und kann damit Transaktionen von diesem Account aus tätigen. Hätte jemand anderes diesen Schlüssel, könnte diese Person von dem Account aus Transaktionen tätigen. Würde man den Schlüssel verlieren, könnte man nicht mehr auf seinen Account und somit auch nicht mehr auf sein Geld zugreifen.
7. a) Nein, da Null kein inverses Element besitzt.

   b) Die reellen Zahlen ohne die Null wären bezüglich der Multiplikation eine Gruppe.

8. Die Ordnung von 9 ist 13 da 9^{13} = 729 und 13*56=728. Kleinere Exponenten ergeben nicht 1.

Kapitel 3: Funktionsweise der Blockchain

9. Würde jemand etwas in einem Block ändern, so würden alle Referenzwerte und somit Hashwerte aller darauffolgenden Blöcke verändert. Die Proof-of-Work Rätsel wären dementsprechend in allen darauffolgenden Blöcken nicht mehr gelöst. Damit die Version des Ledgers trotzdem verbreitet würde, müssten in allen darauffolgenden Blöcken die Proof-of-Work Rätsel gelöst werden und ein neuer Block mit Proof-of-Work Rätsel angehängt werden. Dies müsste geschehen, bevor jemand anders es schafft, einen neuen Block anzuhängen.
10. Miner lösen die Proof-of-Work Rätsel und hängen somit neue Blöcke an die aktuelle Version des Ledgers an. Auch müssen sie sicherstellen, dass alle Transaktionen korrekt sind (Signiert mit DSA und der Sender muss über die nötige Menge an Kryptowährung verfügen).
11. Rechenzentren haben durch die enorme Rechenkapazität gute Chancen, neue Blöcke an die aktuelle Version des Ledgers anzuhängen. Dies kann sich rentieren, da ein Miner beim Anhängen eines neuen Blocks an die aktuelle Version des Ledgers durch Kryptowährung belohnt wird. Dadurch lässt sich Geld verdienen.
12. Es ist theoretisch möglich, seinen Computer zum Minen von Bitcoins zu verwenden und es zu schaffen, als erstes einen neuen Block an die aktuelle Version des Ledgers anzuhängen. Jedoch ist die Wahrscheinlichkeit, dass dies gelingt, enorm gering. Dafür müsste der eigene Computer nämlich nach sehr wenigen Versuchen den passenden Nonce finden. Anderenfalls hätte er keine Chancen gegen die Rechenleistung der Rechenzentren und enorm schnellen Computer im Peer-to-Peer Netzwerk. Es würde also vor allem viel Strom verbraucht und die Chance auf einen Gewinn wäre minim.
13. Mithilfe des Gossip-Prinzips können Informationen im Netzwerk verteilt werden. Ein Computer, der eine Information erhält, sendet diese weiter an seine „Kollegen“. Diese „Kollegen“ senden die Information weiter an ihre Kollegen. Somit wird die Information effizient verteilt.
14. Ein Computer sendet in regelmässigen Abständen eine unwichtige Nachricht (Ping) an alle die Computer auf seiner Liste und erwartet eine Antwort, welche auch aus einer unwichtigen Nachricht (Pong) besteht. Wenn mehrfach keine Antwort folgt, wird der nichtantwortende Computer von der Liste der „Kollegen“ entfernt.
15. Mit den Merkle Trees können viele Transaktionen in einem einzigen Hashwert (Merkle Root) zusammengefasst werden. Jede Änderung bei einer Transaktion ist jedoch sofort im Merkle Root ersichtlich.
16. Er muss genau 7 + 1 also 8 Daten besitzen, um die Merkle Root auszurechnen. Grund hierfür ist, dass $\log_2(128) = 7$ und man muss noch die zu überprüfende Datei besitzen.
17. Man sollte mindestens sechs neue Blöcke abwarten, was etwa einer Stunde entspricht. Erst ab diesem Zeitpunkt sollte man eine Transaktion als getätigt ansehen. Dann ist die Wahrscheinlichkeit nämlich gross genug, dass diese Transaktion in der aktuellen Version des Ledgers enthalten ist.

Kapitel 4: Smart Contracts

18. Der auf der Blockchain abgespeicherte Programmcode kann so geschrieben werden, dass er die programmierten Vorgänge automatisch ausführen kann. Diesen Sachverhalt kann man sich wie einen Verkaufsautomaten vorstellen. Wird eine Münze eingeworfen (eine Bedingung erfüllt), so wird ein Produkt ausgeworfen (ein Vorgang automatisch

durchgeführt). Dadurch, dass der Programmcode auf der Blockchain abgelegt ist, ist es unmöglich, dass dieser im Nachhinein verändert wird oder Variablen verändert werden, sodass betrogen werden könnte.

19. Eigene Lösungsvorschläge

20. ERC20 Tokens entsprechen Kryptowährung, sind also mit Geld vergleichbar. Alle Tokens haben hier dieselben Eigenschaften. Bei ERC721 Tokens haben die einzelnen Tokens unterschiedliche Eigenschaften (beispielsweise haben Kryptokitties ein unterschiedliches Aussehen). Die ERC721 Tokens sind mit Schmuck vergleichbar.

21. Lösungsvorschlag:

```
pragma solidity ^0.4.24; //neuste Version von Solidity verwenden

contract counterContract {

   uint256 counter = 5; //Counter definieren

   function add() public {
     counter++; //Counter um 1 erhöhen
   }

   function subtract() public {
     counter--; //Counter um 1 verringern
   }

   function getCounter() public constant returns (uint256) {
     return counter; //Counter als Ausgabewert ausgeben
   }

}
```