# Ethereum und Smart Contracts

René Müller
Fachbereich Informatik
Hochschule Bonn-Rhein-Sieg
St. Augustin, Deutschland
rene.mueller@smail.inf.h-brs.de

***Zusammenfassung*—Diese Arbeit beschäftigt sich mit dem Blockchain-Netzwerk Ethereum und Smart Contracts, welche auf diesem ausgeführt werden können. Zunächst werden die Kernbestandteile von Ethereum erläutert, wie die Ethereum Virtual Maschine, der Ether und das Thema Gas. Nachdem die Grundlagen geschaffen wurden, geht es im späteren Verlauf der Arbeit hauptsächlich um das Thema Smart Contracts. Fragen, welche hier geklärt werden sollen, sind unter anderem: Was ist ein Smart Contract? Wie funktionieren Smart Contracts in Verbindung mit Ethereum? Welche Aufgaben können mit Smart Contracts gelöst werden? Für welche Aufgaben sind Smart Contracts nicht geeignet? Und wie und unter welchen Bedingungen kann es mit der Technologie in Zukunft weitergehen? Die Blockchain an sich wird in dieser Arbeit nicht ausführlich betrachtet. Nach einem kurzen Einstieg zu Begin wird das Wissen über die Grundlagen der Blockchain-Technologie als bekannt vorausgesetzt. Weiterführende Einleitungen können den Arbeiten *„Blockchain: Funktionsweise und Applikationsmöglichkeiten“* und *„Kryptowährungen“* entnommen werden [1] [2].**

***Tags*—Blockchain, Ethereum, Ether, EVM, Smart Contracts, Solidity, DAO.**

## I. Blockchain

Bei einer Blockchain handelt es sich technisch betrachtet um ein Peer-to-Peer Netzwerk, welches Blöcke einer festen Länge aneinanderreiht. Die Integrität und Vertraulichkeit der einzelnen Blöcke wird hierbei mittels Kryptografie sichergestellt. Das erste Konzept einer Blockchain wurde im Jahre 2008 in dem Paper *„Bitcoin: A Peer-to-Peer Electronic Cash System“* des Autors *Satoshi Nakamoto* vorgestellt. *Satoshi Nakamoto* ist ein Pseudonym, die wahre Identität des Autors ist bis heute ungeklärt. Das Paper spricht zwar noch nicht von einer Blockchain, beschreibt aber Vorgehensweisen, die von den meisten Blockchain-Implementierungen heutzutage genau so eingesetzt werden. Die hauptsächliche Errungenschaft, der Blockchain-Technologie ist es, dass mit dieser das *Double-Spending Problem* in dezentralen Systemen gelöst werden kann [3].

Die kleinste Einheit einer Blockchain ist ein Block. Ein Block besteht neben einem *Identifikator* und weiteren *Metadaten* hauptsächlich aus den *Transaktionsdaten*. Weiterhin besteht ein Block aus einer *Signatur*, einer *Nonce* und vor allem aus der Signatur des Vorgängerblocks. Die Aufgabe bei der Berechnung für den aktuellen Block ist es nun, die Nonce so zu wählen, dass die resultierende Signatur mit einer vorgegebenen Anzahl führender Nullen beginnt. Die so entstandene Signatur wird im nächsten Block mitgespeichert und hat somit neben der Nonce einen Einfluss auf die Signatur des Blocks. Jeder weitere neue Block wird dann genau nach demselben Muster erzeugt. Abbildung I skizziert die Verknüpfung der einzelnen Blöcke [4].

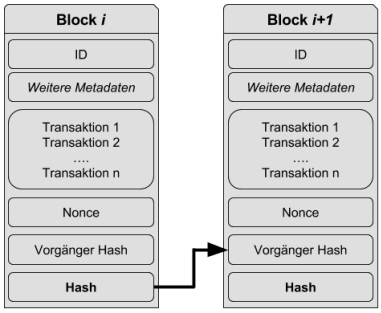


Abbildung 1. Blockchain

Das so entstehende Konstrukt aus aneinandergereihten Blöcken bildet dann die Blockchain. Möchte ein Angreifer nun einen Block in der Blockchain manipulieren, um zum Beispiel eine Transaktion zu manipulieren, würde sich natürlich auch die resultierende Signatur für den manipulierten Block ändern, da die Transaktionsdaten mit in die Signatur eingeflossen sind. Da diese Signatur aber auch in dem Nachfolger Block gespeichert wird, müsste der Angreifer auch für diesen Block die Nonce so ändern, dass die Signatur wieder der Vorgabe entspricht. Diesen Vorgang müsste der Angreifer dann für jeden nachfolge Block wiederholen, bis alle Signaturen wieder konsistent sind. Dies ist nur dann möglich, wenn der Angreifer mehr als 50 Prozent der Rechenkraft im Netzwerk kontrolliert, da sonst mehr neue Blöcke entstehen, als der Angreifer in der gleichen Zeit manipulieren kann. Das beschriebene Konzept wird auch *Proof-of-Work* genannt, auch das Ethereum-Netzwerk nutzt aktuell dieses Verfahren. *Proof-of-Work* steht zurzeit in der Kritik, da der Energiebedarf für dieses Verfahren sehr rasant steigt. Hintergrund ist hierbei die Vorgabe nach der neue Blöcke erstellt werden müssen. Diese wird immer schwieriger, je mehr Rechner sich dem Netzwerk anschließen. Am Anfang, als die Rechenleistung noch sehr gering war, rechte es beispielsweise aus einen Signatur zu berechnen, die nur sehr wenige führende Nullen besitzt. Mit Zunahme der *Miner*, steigt aber auch die Rechenleistung im Netz, sodass immer schwierigere Signaturen berechnet werden müssen (bei Bitcoin sind es beispielsweise aktuell zehn führende

Nullen je Signatur). So entsteht ein Teufelskreis aus immer mehr Hardware, die immer schwierigere Signaturen berechnen muss, was einen gewaltigen Energiebedarf voraussetzt. Aus diesem Grund wird aktuell auch an anderen Konsensverfahren gearbeitet. Eines der Bekannteren ist das sogenannte *Proof-of-Stack* Verfahren. Hierbei werden Transaktionen nicht von Minern bestätigt, sondern sie werden von den Mehrheitshaltern im Netzwerk entschieden. Ethereum möchte in einem späteren Update auf das *Proof-of-Stack* Verfahren umsteigen [5].

## II. Ethereum

Der Bitcoin war die erste Umsetzung der Blockchain-Technologie. Allerdings ist die Bitcoin-Blockchain in ihrer Funktionsweise sehr beschränkt. Im Grunde funktioniert sie wie ein einfacher Stack und kann lediglich dazu verwendet werden, Transaktionen von einem Konto auf ein anders vorzunehmen. Im Fall von Bitcoin, kann man sich die Blockchain also als großes Buch vorstellen, in welchem alle Transaktionen erfasst werden. Ein Block mit Transaktionen repräsentiert dabei eine Seite in diesem Buch. Die Blockchain-Technologie an sich bietet aber ein sehr viel größeres Potenzial, als dass einfache Verschieben von Geldwerten. Dieses wurde bereits im Paper von *Nakamoto* erwähnt [3].

Mit zunehmender Beachtung und dem Erfolg der Bitcoin-Blockchain, entstanden immer mehr *Altcoins* (alternative Coins zum original Bitcoin), welche das grundlegende Konzept der Blockchain aufgriffen und dann sinnvoll erweiterten. Gegen Ende 2013 entstand so dann auch das Ethereum Projekt. Sein Erfinder *Vitalik Buterin* hatte die Idee, eine programmierbare Blockchain zu entwickeln, auf welcher beliebiger Code ausgeführt werden kann. Im Vergleich zu Bitcoin, sollte Ethereum also *turingvollständig* sein und über eine *Statemachine* Zustände festhalten können. Ethereum funktioniert im Kern wie Bitcoin. Transaktionen werden an die Blockchain gesendet und dann von den Minern verarbeitet. Diese Transaktionen besitzen aber anders als im Bitcoin-Netzwerk ein Gedächtnis, sodass Zustände über mehrere Transaktionen hinweg gelesen und verändert werden können. Abhängig von den Zuständen, können sich gleiche Transaktionen dann unterschiedlich verhalten. Zusätzlich ermöglicht es die Turingvollständigkeit, Programmierkonstrukte wie Schleifen zu nutzen. Wie dies in Ethereum genau umgesetzt wurde, klären die nächsten Abschnitte [6].

## III. Ethereum Virtual Machine

Ethereum ist eine programmierbare Blockchain. Anstatt dem Nutzer eine Menge an vorgefertigten Funktionen zur Verfügung zu stellen, soll dieser die Möglichkeit haben, jede beliebige Funktion, die er benötigt, selbst zu erstellen. Im Bitcoin-Netzwerk gibt es beispielsweise die einfache Funktion, eine Transaktion auszuführen. Diese gibt es im Ethereum-Netzwerk zwar auch, der Nutzer kann diese Funktion aber bei Bedarf noch beliebig erweitern. Beispielsweise kann eine Transaktion von Geld an eine Bedingung geknüpft werden. Die Transaktion wird nur dann freigegeben, wenn der Empfänger vorher ein digitales Gut übergeben hat. Andernfalls erfolgt nach einer gewissen Zeit ein automatischer Rollback und der Sender erhält sein Geld zurück. Diese Art von Transaktionen wird auch Smart Contract genannt. Abschnitt VI behandelt Smart Contracts im Detail [7].

Den Kern dieser Blockchain Programme bildet bei Ethereum die *Ethereum Virtual Machine (EVM)*, welche beliebigen Code ausführen kann. Die EVM agiert hierbei ähnlich wie die *Java Virtual Machine (JVM)* im Java Kontext. Sie bildet eine sichere Schnittstelle auf die Ressourcen eines *Knoten (eng. Node)* im Ethereum-Netzwerk. Entwickler können Code in *JavaScript*, *Python* oder anderen Programmiersprachen schreiben und mittels Compiler in Maschinencode für die EVM kompilieren. Die EVM führ dann den Code auf den Nodes aus. Auch wenn es mit einem entsprechenden Compiler im Prinzip möglich ist, Code für die EVM in vielen bekannten Programmiersprachen zu erstellen, hat sich jedoch für die Programmierung von Programmen auf der Ethereum-Blockchain Solidity durchgesetzt. Solidity wird im Abschnitt VII genauer behandelt [8].

Die EVM hat aber noch einen weiteren Nutzen. Nachdem der Code per Transaktion im Netzwerk angekommen ist, wird dieser zunächst von dem Knoten ausgeführt, der ihn in seinem Block mit aufnimmt. Ist dieser Block dann einmal in der Blockchain, sorgt die EVM auch dafür, dass derselbe Code auf jedem anderen Knoten ausgeführt wird und die Zustände auf jedem Knoten somit immer konsistent sind. Wird auf einem Computer ein neuer Knoten erstellt, befindet sich dieser zunächst in einem Initialzustand. Anschließend verbindet sich der neue Knoten mit bestehenden Knoten im Netzwerk und es wird damit begonnen, die Blockchain herunterzuladen. Dies geschieht Block für Block, und zwar in genau der Sequenz, wie die Blockchain entstanden ist. Alle Programmaufrufe, die sich dabei in einem Block befinden, werden auf dem neuen Knoten genau so aufgerufen, wie diese schon auf allen andern bereits existierenden Knoten ausgeführt worden sind. Mit jedem Aufruf ändert sich dabei dann auch der Zustand des neuen Knotens. Dies geschieht so lange, bis der neue Knoten auf genau demselben Stande ist, wie alle andern Knoten im Netzwerk. Auf diese Weise wird sichergestellt, dass ein Programm welches einmal in die Blockchain geladen wurde, auf jeden Fall auch ausgeführt wird, da es unaufhaltsam auf jedem Knoten ausgeführt wird [8].

Durch diese massive Parallelität werden die Berechnungen im Ethereum-Netzwerk keines Falls effizienter. Im Gegenteil, Transaktionen im Ethereum-Netzwerk sind sogar um ein vielfaches langsamer und um ein vielfaches kostenintensiver, als in herkömmlichen Computersystemen. Aber dadurch, dass jeder Knoten im Netzwerk die EVM laufen lässt, entsteht ein maximaler Konsens. Durch diesen dezentralen Konsens bietet

das Ethereum-Netzwerk ein Maximum an Betrugssicherheit. Es sorgt auch dafür, dass es keine Ausfallzeit gibt (außer alle Knoten fallen aus) und dass Daten, die einmal in der Blockchain gespeichert sind, änderungsresistent und frei von Zensur sind. Aus diesen Gründen wird Ethereum auch der *Weltcomputer* genannt, da die EVM dafür sorgt, dass man das gesamte Ethereum-Netzwerk als einen Computer sehen kann [7].

Wie bei jeder anderen Entwicklungsumgebung ist es dem Programmierer überlassen, was er aus dem Werkzeug macht, welches ihm zur Verfügung gestellt wird. Einige Anwendungen können aber besonders von Ethereum profitieren. Hierzu zählen beispielsweise klassische Anwendungsfälle aus der IT-Sicherheit. In der IT-Sicherheit wird meistens ein Maximum an Verschwiegenheit, Integrität und Verfügbarkeit gefordert. Für die Erfüllung dieser Anforderungen ist das Ethereum-Netzwerk durch seine Eigenschaften besonders prädestiniert. Es gibt allerdings auch Anwendungsfälle, bei denen der Einsatz von Ethereum keinen Sinn macht und eher kontraproduktiv wirken kann. Spielt die Intigriät von Daten beispielsweise keine besondere Rolle, da diese nur temporär benötigt werden, machen klassische Datenbanksysteme weit aus mehr Sinn, da diese sehr viel effizienter arbeiten können [7].

## IV. Ether

Der Ether ist die Währung im Ethereum-Netzwerk. Anders als beim Bitcoin ist sein Nutzen aber nicht rein als Währung gedacht, sondern soll hauptsächlich als Gebühr für Berechnungen auf der EVM dienen. Der Ether ist die größte Einheit bei Ehtereum. Ähnlich wie bei *Euro/Cent*, gibt es aber auch noch eine kleinere Einheit, welche einen Teilwert eines Ethers repräsentiert. Dieser heißt im Ehtereum-Netzwerk Wei. $10^{18}$ Wei entspricht hierbei einem Ether. Zwischen einem Wei und einem Ether gibt es in Tausenderschritten jeweils eine weitere Einheit. Diese sind in Tabelle I zu sehen. Die Namen sind an Persönlichkeiten aus dem Bereich Informatik und im speziellen der Kryptografie angelehnt [7].

| **Einheit** | **Wei Wert** | **In Wei** |
|---|---|---|
| wei | 1 wei | 1 |
| Kwei (babbage) | 1e3 wei | 1.000 |
| Mwei (lovelace) | 1e6 wei | 1.000.000 |
| Gwei (shannon) | 1e9 wei | 1.000.000.000 |
| microether (szabo) | 1e12 wei | 1.000.000.000.000 |
| milliether (finney) | 1e15 wei | 1.000.000.000.000.000 |
| ether | 1e18 wei | 1.000.000.000.000.000.000 |

Tabelle I
Ether Einheiten [9]

Ether kann auf verschiedene Wege gewonnen werden. Zum einen kann jeder im Ethereum-Netzwerk Ether Minen, indem er Blöcke berechnet und diese dann zur Blockchain hinzufügt. Zum anderen kann Ether auf bekannten Kryptobörsen im Internet gekauft werden. Eine dritte Möglichkeit an Ether zu kommen, ist die Erstellung eines Smart Contracts. Über einen solchen Vertrag kann ein Kunde beispielsweise eine Ware kaufen. Smart Contracts werden im Abschnitt VI genauer betrachtet [9].

Um den so erhaltenen Ether dann aufzubewahren, wird ein Wallet benötigt. Vereinfacht dargestellt ist ein Wallet, im Kontext von Blockchains, ein Programm, welches es einem erlaubt, über einen privaten Schlüssel, sein Guthaben abzufragen und Transaktionen im Netzwerk zu tätigen. Dabei wird zwischen Software-Wallets und Hardware-Wallets unterschieden. Hardware-Wallets sind vergleichbar mit Smartcards, welche den privaten Schlüssel verborgen halten und über eine sichere Schnittstelle eine Interaktion mit diesem ermöglichen. Bei der Wahl des Wallets sollte man sich möglichst nicht auf Drittanbieter verlassen, welche den privaten Schlüssel für den Anwender verwalten. Dies kann zwar auf den ersten Blick komfortabler sein, birgt aber auch die Gefahr, dass der Drittanbieter den Schlüssel verliert oder missbraucht. In der Vergangenheit gab es bereits mehrere teils größere Anbieter dieser Art, die durch einen Hackerangriff die Coins ihrer Kunden verloren haben [2].

Eine der besten Art sein Ether aufzubewahren, ist das offiziell Ethereum-Wallet, welches von der Ethereum Foundation entwickelt und zur Verfügung gestellt wird. Mit diesem hat der Nutzer die volle Kontrolle über seine Schlüssel und kann über eine grafische Benutzeroberfläche Transaktionen durchführen und weitere Einstellungen vornehmen [10].

## V. Gas

Gas ist ein Mechanismus von Ethereum, um das Netzwerk lauffähig zu halten. Wie bei jedem anderen Computersystem wirkt auch bei der EVM das Halteproblem aus der theoretischen Informatik. Es kann nicht mit Sicherheit bestimmt werden, ob ein Programm auch wirklich bei einer beliebigen Eingabe hält. Ein kleiner Programmierfehler, kann bereits zu einer Endlosschleife führe, welche einmal im Netzwerk angekommen, auf jedem Knoten ausgeführt wird. Dies würde natürlich zur Folge haben, dass das gesamte Netzwerk zum Erliegen kommt. Aber nicht nur Programmierfehler werden mit Gas adressiert, auch mögliche Denial-of-Service (DoS) Attacken auf das Netzwerk sollen so teuer gemacht werden [7].

*Gas* funktioniert so, dass jede Transaktion im Netzwerk mit einer Gebühr verbunden ist. Die EVM rechnet dabei sehr genau ab, in dem jede Operation im Programm gezählt wird. Wird beispielsweise eine Schleife mit hundert Iterationsschritten aufgerufen und im Schleifenkörper befinden sich zwei Operationen, die jeweils eine Zahl inkrementieren, dann zählt die EVM hierfür 200 Anweisungen. Eine Operation, die in etwa *32 bit* an Speicher benötigt, entspricht hierbei in etwa ein Gas. Soll nun eine Transaktion im Netzwerk durchgeführt werden, muss der Initiator dieser Transaktion, ein sogenanntes *Gaslimit* angeben. Dies ist die maximale Anzahl an Gas, die für die auszuführende Transaktion zur Verfügung steht. Sollte die Transaktion nun in eine Endlosschleife gelangen, wird diese mit erreichen

des Gaslimits automatisch beendet und alle Änderungen am Zustand des Netzwerks durch ein Rollback zurückgesetzt. Das aufgewendete Gas wird dabei nicht zurück erstattet. Anders sieht es aus, wenn das Gaslimit zu hoch angesetzt wurde und die Transaktion erfolgreich ausgeführt wurde. In diesem Fall wird die Differenz erstattet [7].

Der Fall, in dem das Gaslimit überschritten wird, kann allerdings zu einem Problem führen. Weiß derjenige, der die Transaktion startet, nicht wie viel Gas diese benötigt, kann dies dazu führen, dass er das Gaslimit zu niedrig setzt. Dies könnte auch bei einem korrekt programmierten Programm der Fall sein, wenn dieses eben nur sehr viele Operationen benötigt. Wird das Gaslimit nun zu niedrig gewählt, erreicht die Transaktion das Gaslimit, bevor das Programm mit seiner Berechnung fertig ist und es erfolgt ein Rollback. Natürlich steht es jedem offen sich die Funktionsweise eines Programms anzuschauen, da der Code offen im Netzwerk liegt und daraus das optimale Gaslimit abzuleiten. Aber in der Praxis würde dies wohl trotzdem häufig zu Problemen führen. Aus diesem Grund geben die Entwickler von Programmen auf der Blockchain meistens mit an, welches Gaslimit bei einem Aufruf ihres Vertrages gewählt werden sollte [7].

Neben dem Gaslimit muss auch noch ein *Gaspreis* für eine Transaktion gesetzt werden. Der Gaspreis gibt an, wie viel Ether der Aufrufende bereit ist, je Gas zu bezahlen. Aktuell liegt der Gaspreis für eine einfache Transaktion von Ether auf ein anderes Konto, bei 21 Gwei (siehe Tabelle I). Um Ether bei der Transaktion zu sparen, kann der Gaspreis natürlich auch niedriger angesetzt werden. Sollte dieser aber zu niedrig ausfallen, ist es für die Miner nicht mehr attraktiv diese Transaktion mit in Ihren Block aufzunehmen und die Transaktion wird nie ausgeführt. Wird der Gaspreis höher angesetzt, ist es für die Miner entsprechend sehr attraktiv die Transaktion mit in ihren aktuellen Block aufzunehmen und die Transaktion wird schneller ausgeführt. Mit dem Gaspreis kann also festgelegt werden, wie schnell eine Transaktion ausgeführt werden soll. Handelt es sich um eine wichtige Transaktion, wird ein hoher Gaspreis gewählt und bei unwichtigen Transaktionen, kann auch ein geringerer Gaspreis gewählt werden. Die komplette Gebühr, je Transaktion, errechnet sich aus folgender Formel [7].

$$Gebühr = Gaspreis \cdot Gaslimit \tag{1}$$

Auf den ersten Blick sieht es etwas umständlich aus, dass hier Gas als extra Einheit genutzt wird, anstatt die Abrechnung je Transaktion direkt in Ether durchzuführen. Der Grund liegt darin, dass die Macher von Ethereum sicherstellen wollten, dass Transaktionen immer im Preis konstant bleiben. Ether ist, wie jede andere Kryptowährung, sehr volatil in ihrem Wert. Alleine im Jahr 2017 gab es Kursschwankungen von mehreren 100$. Der Aufruf derselben Funktion könnte sich somit an zwei aufeinanderfolgenden Tagen, um ein Vielfaches verteuern (oder billiger werden). Aus diesem Grund wurde das Gas geschaffen. Der Gaspreis wird also immer an den aktuellen Ether Kurs angepasst. Ist der Kurs hoch genug, werden die Miner trotz eines geringen Gaspreises bereit sein, die Transaktion mit in ihren Block aufzunehmen und sollte der Kurs sinken, muss entsprechend ein höherer Gaspreis je Transaktion gewählt werden [7].

## VI. Smart Contracts

### A. Entstehung

Die Idee von Smart Contracts ist bereits bedeutend älter, als die Blockchain-Technologie. Bereits 1997 wurde das Konzept von Smart Contracts in dem Artikel *„The Idea of Smart Contracts“* grundlegend erläutert. Der Autor hat sich angeschaut, wie Verträge grundlegend ablaufen und welche Probleme es dabei gibt. Demnach sollen Verträge im besten Fall so ablaufen, wie ein Automat abläuft. Der Kunde möchte ein Produkt in dem Automaten kaufen und bekommt daraufhin den Preis für dieses Produkt angezeigt. Anschließend wirft der Kunde Münzen in den Automaten und bekommt dann, abhängig vom Zustand des Automaten das Produkt ausgegeben oder seine Münzen zurück. Der Vertrag erfüllt sich also selbst und es ist zu jedem Zeitpunkt transparent, was geschieht. Dies hat einige Vorteile im Vergleich zu anderen Vertragsformen. Zum einen kann ein solcher Vertrag, mit dem Automaten, von jedem ausgeführt werden, der genügend Münzen besitzt. Ein Schloss an dem Automaten verhindert, dass unbefugte den Zustand des Automaten ändern oder Münzen aus dem Automaten entwenden können (es wird angenommen, dass der Betrag in dem Automaten so geringer ist, dass der Aufwand das Schloss zu knacken sich nicht lohnt) [11].

Wird das Automatenmodel auf alltägliche Verträge angewendet, können so viele bekannte Probleme gelöst werden. Beispielsweise ist es bei den meisten Verträgen üblich, dass ein Mittelsmann benötigt wird. Möchte ein Autohändler beispielsweise ein Auto verkaufen, welches in Raten abgezahlt werden soll, so benötigt er in der Regel ein Inkassounternehmen, um sein Fahrzeug wieder zu bekommen, sollte der Käufer seiner Pflicht nicht nachkommen, die Raten zu bezahlen. Mithilfe eins Smart Contract könnte in dem Fahrzeug ein Schloss installiert werden, welches die Kontrolle über das Schloss an den Käufer übergibt, sobald eine initiale Zahlung an den Smart Contract überwiesen wurde. Solange monatlich die Zahlungen in den Smart Contract eingehen, bleibt die Kontrolle auch beim Käufer des Autos. Sollte dieser sich allerdings entscheiden, keine Raten mehr zu bezahlen, so würde die Kontrolle automatisch an den Autohändler zurückgehen, ohne jegliches Zutun seinerseits. Da so der Mittelsmann entfällt (in der Praxis die Mittelsmänner, da der Kauf eines Autos in der Regel noch weitaus mehr Parteien beinhalten), hat der Käufer auch etwas davon, da das Fahrzeug für ihn so höchst wahrscheinlich Günstiger zu kaufen sein wird. Dieses Prinzip lässt sich nicht nur beim Autokauf anwenden, sondern auch in vielen anderen Bereichen des täglichen Lebens. Soll etwas online gekauft

werden, wird ein Mittelsmann in Form eines Auktionshauses benötigt, Eigentumsrechte müssen von Notaren beglaubigt werden und vieles mehr [11].

Es gibt noch ein weiteres großes Problem, welches von Smart Contracts gelöst werden kann. Im obigen Beispiel wir davon ausgegangen, dass die Kontrolle über das Auto direkt nach dem ersten Ausbleiben einer Zahlung, an den Autohändler zurückgeht. In der Praxis wäre es wohl für beide Parteien vorteilhafter, wenn dies erst nach zwei oder gar mehreren ausbleibenden Zahlungen geschieht. In einem Smart Contract kann auch dies berücksichtigt werden. Es kann ganz granular festgelegt werden, unter welchen Bedingungen die Besitzrechte an dem Fahrzeug geändert werden sollen. Auf diese Weise können dann auch teure Rechtsstreitigkeiten vermieden werden. Die meisten Rechtsstreitigkeiten im Zivilrecht gehen wohl darauf zurück, dass die jeweiligen Parteien einen Vertrag zu ihren Gunsten auslegen möchten. Ein Smart Contract hingegen hat den Vorteil, dass dessen Ausführung nicht interpretierbar ist, sondern immer deterministisch. Dank der Blockchain sogar immer verfügbar und fälschungssicher [11].

Smart Contracts lassen sich überall dort am besten anwenden, wo es möglich ist, einen Gegenstand von Wert elektronisch abzubilden. Alles, was hierüber hinausgeht, kann von einem Smart Contract nicht abgedeckt werden. So können beispielsweise die Besitzrechte an einem teuren Edelstein mittels Smart Contract eindeutig festgestellt werden, sollte dieser aber gestohlen werden, bringt ein Smart Contract diesen auch nicht wieder zurück. Ein weiteres Problem kann sein, dass ein Smart Contract bestimmte Aspekte eins Vertrags nicht abdeckt. Sollte das Auto aus dem Beispiel nach einem Jahr an den Autohändler zurück gehen, weil der Käufer seine Raten nicht bezahlt hat, ist es wichtig, dass der Smart Contract auch genau behandelt, wie sich der bisher gezahlte Betrag mit der Wertminderung des Autos berechnet und wie viel Geld der Käufer aus dem bezahlten Betrag wieder zurück erhält. Sollte dies nicht geregelt sein, wird solch ein Fall wohl auch in Zukunft wieder vor Gericht landen [11].

### *B. Smart Contracts in Ethereum*

Wer am Ethereum-Netzwerk teilnehmen möchte, muss, wie im Bitcoin-Netzwerk auch, zunächst ein Nutzerkonto erstellen. Dieses Nutzerkonto wird auch *Externally Owned Accounts (EOA)* genannt. Ein EOA wird über den privaten Schlüssel kontrolliert. Dieser Schlüssel ist alles, was zwischen dem Konto und der Außenwelt steht. Anders als Bitcoin unterscheidet Ethereum aber zwischen zwei Arten von Accounts. Neben den oben genannten EOA Konten gibt es noch sogenannte Contract Accounts. Ein Contract Account ist im Grunde ein Programm, welches gewisse Funktionen nach außen anbietet, die dann von EOA Konten oder auch anderen Contract Accounts aufgerufen werden können. Initialisiert werden Contract Accounts aber immer von EOA Konten. Der Entwickler eines Contract Accounts kann bei dessen Erstellung festlegen, wie viel Kontrolle er über den Contract Account behalten möchte. Auch die Abgabe der gesamten Kontrolle über einen Contract Account ist möglich, sodass dieser sich komplett selbst verwaltet. Über die Sichtbarkeit der Funktionen kann der Ersteller auch festlegen, wer mit dem Contract Account interagieren darf und in welchem Rahmen. Eine Interaktion geschieht immer mittels einer Transaktion. Soll beispielsweise eine Funktion eines Contract Accounts aufgerufen werden, muss eine Transaktion an den Contract Account gesendet werden. Ein Contract Account entspricht bei Ethereum hierbei einem Smart Contract. Wie Smart Contracts genau aufgebaut sind und wie diese in Ethereum umgesetzt werden, beschreibt der nächste Abschnitt [12].

Abbildung 2 zeigt den kompletten Zusammenhang, wie ein Smart Contract in Ethereum in die Blockchain gelangt. Zunächst wird dieser in einer Programmiersprache geschrieben (hier Solidity). Anschließend wird das so entstandene Programm zu Bytecode kompiliert, welcher dann auf der EVM jedes Knoten im Netzwerk ausgeführt wird.

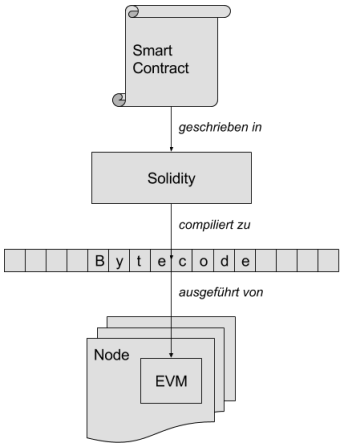


Abbildung 2. Smart Contract

## VII. SOLIDITY

Solidity ist die empfohlene Programmiersprache, um Smart Contracts für die Ethereum-Blockchain zu entwickeln. Solidity wurde von der Ethereum Fondation entwickelt und wird von dieser auch vorangetrieben. Die Sprache orientiert sich an Konzepten bekannten Programmiersprachen wie *C++*, *Python* und *JavaScript*. Mithilfe von Solidity kann Quellcode geschrieben werden, welcher dann von der EVM interpretiert werden kann. Die Syntax orientiert sich an Java, hat aber auch einige untypische Eigenheiten. Neben den normalen Schlüsselwörtern für Bedingungen und Schleifen, die sich weitestgehend an bekannten Programmiersprachen orientieren, gibt es in Solidity auch sehr viele Schlüsselwörter, die Blockchain spezifisch sind. In der Regel wird beim Programmieren ein Ereignis in der Blockchain angestoßen oder es wird auf ein Ereignis aus der Blockchain gewartet und dann auf dieses reagiert. Aus diesem Grund wird Solidity auch die erste *Contract-Orientierte Programmiersprache*

genannt [7].

Einer der häufigsten Einsatzzwecke für Smart Contracts ist zurzeit, das Erstellen von Token. Ein Token ist eine Währung, basieren auf der Blockchain. Welchen Gegenwert die Währung besitzt, kann der Ersteller über den Smart Contract dabei selbst festlegen. Dies kann theoretisch ein beliebiges digitales Gut sein. In der Regel werden Token aber zunächst gegen Ether ausgegeben. Möchte jemand an einem Tokenverkauf teilnehmen, muss er lediglich Ether an einen Smart Contract schicken und erhält dann im Gegenzug eine entsprechende Anzahl an Tokens. Dieses Verfahren ist auch unter dem Namen *Initial Coin Offerings (ICO)* bekannt geworden. Soll beispielsweise eine neue App entwickelt werden, kann ein Unternehmen dadurch Geld einnehmen, dass es zunächst mithilfe eines Smart Contracts Tokens gegen Ether verkauft (welches wiederum in Euro oder Doller gewechselt werden kann). Im Gegenzug stellen die verkauften Token in der fertigen App einen Wert dar, welchen die Investoren dann nutzen können oder mit einem Gewinn an spätere Nutzer weiterverkaufen können [7].

### A. ERC-20

Doch wie können die neu erstellten Token von deren Besitzern eingesehen werden? Da es sich im Prinzip nur um Daten in der Blockchain handelt, können die Token theoretisch von jedem mit einem Knoten eingesehen werden. Aber es wäre wünschenswert, dass diese Token auch über das Wallet einsehbar sind und dort mit dem Ether zusammen verwaltet und gehandelt werden können. Um dies zu erreichen, gibt es den *ERC-20 Standard*. Hierbei handelt es sich um Funktionen und Ereignisse, die ein Smart Contract implementieren muss, damit dessen Token in einem Wallet gelistet und von diesem gehandelt werden können. Im Grunde wird also ein Interface spezifiziert, welches eine einheitliche Nutzung aller Token ermöglicht. Listing 1 zeigt, welche Methoden implementiert werden müssen [7].

```
contract ERC20Interface {
    function totalSupply() public constant returns (
        uint);

    function balanceOf(address tokenOwner) public
        constant returns (uint balance);

    function allowance(address tokenOwner, address
        spender) public constant returns (uint
        remaining);

    function transfer(address to, uint tokens)
        public returns (bool success);

    function approve(address spender, uint tokens)
        public returns (bool success);

    function transferFrom(address from, address to,
        uint tokens) public returns (bool success);


    event Transfer(address indexed from, address
        indexed to, uint tokens);

    event Approval(address indexed tokenOwner,
        address indexed spender, uint tokens);
}
```

Listing 1. ERC-20 Interface [13]

Die wichtigsten Funktionen sind hierbei `balanceOf()` und `transfer()`, da über diese Token versendet werden können und erfragt werden kann, wie viele Token ein Konto besitzt. Über das Ereignis Transfer informiert der Vertrag, dass ein Transfer stattgefunden hat. Weiterhin müssen für einen Tokenvertrag noch die drei Attribute `name`, `symbol` und `decimals` öffentlich einsehbar (`public`) gesetzt werden. Die Attribute `name` und `symbol` sind selbst erklärend und das Attribut `decimals` gibt an, wie viele Dezimalstellen das Token maximal besitzt. Die weiteren Methoden sollten für einen offiziellen Tokenverkauf auch implementiert werden, da viele Börsen es ablehnen sonst mit diesem Token zu handeln. Diese restlichen Methoden erlauben es Dritten, beispielsweise den Börse, Token für den Nutzer zu überweisen. Hierzu wird im Smart Contract ein Array angelegt, in welchem verwaltet wird, ob ein Nutzer die Erlaubnis für eine Überweisung an einen anderen Nutzer erteilt hat (siehe Listing 2) [13].

```
// Owner of account approves the transfer of an
    amount to another account
mapping(address=>mapping (address=>uint)) allowed;
```

Listing 2. Allowance [13]

Diese Erlaubnis kann dann über die Methode `approve()` vom Besitzer der Token erteilt werden, wodurch diese im Array `allowed` gespeichert wird. Es wird sowohl gespeichert, an welche Adresse überwiesen werden darf, als auch wie viele Token überwiesen werden dürfen. Über die Funktion `allowance()` kann diese Informationen abgefragt werden. Wurde eine solche Erlaubnis erteilt, kann über die Methode `transferFrom()` eine Überweisung von einem Dritten ausgeführt werden [13].

### B. Hbrs Token

Im Listing 3 ist ein Smart Contract zu sehen, welcher einen Tokenverkauf ermöglichen könnte. Dieses Token könnte beispielsweise von Studenten der Hochschule Bonn-Rhein-Sieg gekauft werden. Würde die vorhanden Infrastruktur der Hochschule angepasst werden, könnten die Studenten dann digitale Leistungen mit diesem Token bezahlen, wie beispielsweise das Anfertigen von Kopien oder das Bezahlen des Essens in der Mensa [14].

```
pragma solidity ^0.4.19;

contract HbrsToken {
    // Events for this contract
    event Transfer(address indexed from, address
        indexed to, uint tokens);

    // Attributes for this contract
    address private owner;
    string  public  constant name = "Hbrs Token";
    string  public  constant symbol = "HBRS";
    uint8   public  constant decimals = 0;
    uint    private constant hardcap =  50 *
        (10**18);
    // Balances for each account
    mapping (address => uint) private tokens;
```

```
    // Modify for functions
    modifier onlyBy(address _account) {
        require(msg.sender == _account);
        _;
    }

    // Constructor
    function HbrsToken() public {
        owner = msg.sender;
    }

    // Show the balance of this contract in ehter
    function getBalance() public onlyBy(owner) view
        returns(uint balance) {
        return this.balance;
    }

    // Buy some tokens
    function buyTokens() public payable returns(bool
         success) {
        require(this.balance +msg.value <= hardcap);
        tokens[msg.sender] += (msg.value / 10**18) *
             1000;
        return true;
    }

    // Fallback for ether sendet to this contract
    function () public payable {
        buyTokens();
    }

    // Get the token balance for account tokenOwner
    function balanceOf(address _tokenOwner) public
        constant returns (uint token) {
        return tokens[_tokenOwner];
    }

    // Transfer tokens to another account
    function transfer(address _to, uint _tokens)
        public returns(bool success) {
        require(tokens[msg.sender] >= _tokens);
        tokens[msg.sender] -= _tokens;
        tokens[_to] += _tokens;
        Transfer(msg.sender, _to, _tokens);
        return true;
    }
}
```

Listing 3. Hrbs-Token [14]

Im Folgenden wird der Code aus dem Listing 3 genauer erläutert. Hierbei wird die Funktionsweise an sich beschrieben und es wird auf Besonderheiten von Solidity eingegangen. Als Referenzsprache wird hierbei Java verwendet, da viele Konzepte sehr ähnlich sind. Der erste Schritt in jeder Solidity Anwendung ist es, die Version des Compilers anzugeben. Dies ist wichtig, da die Sprache sich ständig weiterentwickelt und immer neue Befehle hinzukommen oder alte entfernt werden. Ist die Compiler Version gesetzt, kann mit der Entwicklung des Smart Contracts begonnen werden. Ein Smart Contract wird in Solidity immer von dem Schlüsselwort `contract` eingeleitet. Er entspricht in etwa dem Schlüsselwort `class` in Java. Tatsächlich ist ein Vertrag in Solidity auch weitergehend so aufgebaut, wie eine Klasse in Java. Es ist beispielsweise auch möglich, aus einem anderen Contract heraus einen Contract mittels Schlüsselwort `new` zu referenzieren und dann Funktionen von diesem aufzurufen. Der große Unterschied ist nur, während in Java mit dem Schlüsselwort `new` jedes Mal eine neue Klasse erstellt wird, handelt es sich bei einem Contract in Solidity immer um ein Singleton [14].

Das Erste, was innerhalb des Vertrags gemacht wird, ist, dass ein Ereignis mit dem Namen `Transfer` erstellt wird. Ereignisse können per Remote Procedure Calls (RPC) von anderen Anwendungen außerhalb der Blockchain abgefragt werden, sodass diese auf Ereignisse in der Blockchain reagieren können. Die Blockchain ist eine Backendtechnologie, mit welcher Endbenutzer in der Regel nicht interagieren wollen und sollen. Der Endbenutzer möchte seine Geschäfte weiterhin über Webinterfaces und Apps erledigen, lediglich die produzierten Daten sollen nicht mehr auf zentralen Datenbanken gespeichert werden, sondern in der Blockchain. Die Ethereum Foundation hat dazu auch eine eigene JavaScript Bibliothek entwickelt, welche Entwicklern hilft, RCPs aus der Blockchain zu beobachten, auf diese zu reagieren und Daten in die Blockchain zu schreibe. Das in diesem Beispiel erstellte Ereignis gibt Auskunft darüber, wer Token versendet hat, an wenn diese Token gesendet wurden und um wie viele Token es sich handelt [14].

Als Nächstes werden die für den Vertrag benötigten Attribute erzeugt. Diese Attribute sind letztlich die Daten, die in der Blockchain gespeichert werden. Dies bedeutet auch, dass jeder der einen Ethereum Knoten betreibt, diese Daten einsehen kann. Aktuell gibt es keinen Mechanismus, einzelne Attribute nicht sichtbar zu machen. Die einzige Möglichkeit, die einem als Programmierer bleibt, ist es Verschlüsslung einzusetzen, sollen Daten nicht öffentlich einsehbar sein. Das erste Attribut soll den Ersteller des Vertrags speichern. Da jeder Nutzer in Ethereum durch seinen öffentlichen Schlüssel eindeutig identifizierbar ist, genügt es, einfach diesen Schlüssel zu speichern. Aus diesem Grund bietet die Sprache auch den Datentyp `address` an, in welchem genau ein Schlüssel gespeichert werden kann. Es gilt zu beachten, dass der Sichtbarkeitsmodifikator im Vergleich zu Java, nach dem Datentyp angegeben wird. Wie schon erwähnt, schützen diese Modifikatoren nicht davor, dass der Zustand der Variable ausgelesen werden kann. Jedem Betreiber eines Knoten ist es möglich, die Variable `owner` zu lesen. Die nächsten drei Variablen sind die zuvor erwähnten Variablen, die für ein Token nach ERC-20 gesetzt werden müssen. Das Attribut `tokens` ist das Herzstück dieses Vertrags. Hierbei handelt es sich im Grunde nur um ein Datenstruktur, welche ein Mapping zwischen einem öffentlichen Schlüssel und einem Integer herstellt. Der Integer gibt dabei an, wie viele Token der Nutzer mit dem zugehörigen Schlüssel besitzt. Auch diese Variable ist öffentlich in der Blockchain einsehbar. Somit kann zu jederzeit verfolgt werden, wie viele Token zu einer bestimmten Adresse gehören [14].

In der nächsten Zeile wird ein sogenannter `modifier` erstellt. Eine Funktion die `public` deklariert wird, kann zunächst von jedem aufgerufen werden, der im Besitz der

Adresse des Smart Contracts ist. Soll aber beispielsweise eine Funktion in den Vertrag eingebaut werden, die nur der Ersteller aufrufen darf, kann dies mithilfe eines `modifier` justiert werden [14].

Als Nächstes wird der Konstruktor des Vertrags erstellt. Wie eine Java-Klasse besitzt auch ein Smart Contract einen Konstruktor, der nur einmalig aufgerufen wird, und zwar wenn der Smart Contract initial in die Blockchain geschrieben wird. Wie alles andere geschieht auch das Erstellen eines Smart Contracts bei Ethereum mittels Transaktion. Der Ersteller des Vertrags schickt eine Transaktion ins Netzwerk, an welcher der Bytecode des Vertrags angehängt wird. Genau bei dieser einen Transaktion wird dann der Konstruktor aufgerufen. Innerhalb des Konstruktors wird hier der Besitzer des Vertrags festgelegt. Mithilfe des `msg` Objekts können in Solidity Informationen über den Vertrag selbst abgerufen werden. `msg.sender` gibt hierbei die Adresse desjenigen zurück, der den Vertrag aufgerufen hat. Da dies innerhalb des Konstruktors der Besitzer selbst ist, kann diese Information in der Variable `owner` gespeichert werden. Es gilt zu beachten, dass der Besitzer in diesem Vertrag nicht mehr geändert werden kann. Soll es die Möglichkeit geben den Besitzer auszutauschen, müsste hierfür eine entsprechende Routine in den Vertrag eingebaut werden [14].

Die Funktion `getBalance()` gibt die Menge an Ether zurück, die auf dem Vertrag gespeichert ist. Der Rückgabewert wird in Solidity mit dem Schlüsselwort `returns` angegeben. Außerdem ist es in Solidity möglich, anzugeben, was eine Funktion mit den Daten im Vertrag macht. Wird das Schlüsselwort `view` verwendet, gibt dies an, dass nur lesend auf Attribute zugegriffen wird. Weiterhin gilt es zu beachten, dass diese Funktion den zuvor definierten Modifier `onlyBy()` verwendet. Diesem wird die Variable `owner` übergeben, welche im Konstruktor mit der Adresse des Erstellers initialisiert wurde. Dies führt dazu, dass eine Transaktion welche diese Funktion aufruft, nur dann erfolgreich ist, wenn sie vom Ersteller des Vertrags ausgeführt wird [14].

`buyToken()` kann von jeder Adresse aufgerufen werden. Über diese Funktion können Nutzer Token kaufen. Das Schlüsselwort `payable` gibt an, dass Nutzer an diese Funktion Ether senden können. Wird Ether an Funktionen ohne dieses Schlüsselwort gesendet, schlägt die Transaktion einfach fehl. Das Erste, was in der Funktion geprüft wird, ist, ob der sogenannte Hardcap schon erreicht ist. Bei einem Tokenverkauf wird in der Regel eine obere Schranke definiert, welche die Anzahl der Token die ausgegeben werden limitiert. Würde dies nicht geschehen, könnten immer mehr Token erstellt werden, was zu einem späteren Zeitpunkt zu einer Inflation führen könnte. Der Hardcap in diesem Beispiel liegt exemplarisch bei 50 Ether. Mithilfe von `msg.value` kann überprüft werden, wie viel Ether an den Vertrag gesendet wurde. Der Adresse des Senders werden dann einfach je gesendetem Ether tausend Token gutgeschrieben [14].

Die Funktion ohne Namen ist bei einem Smart Contract in Solidity immer die Funktion, die aufgerufen wird, sollte jemand den Vertrag ohne Angabe einer Funktion aufrufen oder eine Funktion aufrufen, die es nicht gibt. Die Funktion wird auch Fallback Funktion genannt. Ohne diese Funktion wäre das Ether das der Aufrufende sendet verloren. In diesem Beispiel wird in der Fallback Funktion einfach die `buyToken()` Funktion aufgerufen [14].

Die letzten beiden Funktionen sind die Implementierungen der Funktionen aus Listing 1. Die Funktion `balanceOf()` gibt die Anzahl der Token für die übergebene Adresse aus der Datenstruktur `token` zurück und mit der Funktion `transfer()` können Token zwischen zwei Nutzer überwiesen werden. Hierbei wird zunächst geprüft ob der Nutzer über genügend Token verfügt, die er versenden möchte. Da über `msg.sender` immer derjenige ermittelt werden kann, der die Transaktion ausgeführt hat, benötigt die Funktion auch nur die Empfänger Adresse. Abschließend wird noch ein Ereignis ausgelöst, welches über die erfolgreiche Überweisung informiert [14].

## VIII. Decentralized Autonomous Organization

Der Kerngedanke einer Blockchain ist es, eine dezentrale Einheit zu schaffen, in welcher niemand die Kontrolle besitzt oder diese übernehmen kann. Für die Daten, die so in der Blockchain gespeichert werden, funktioniert dies auch recht gut, aber eine entscheidende Schwachstelle gibt es trotzdem. In der Regel gibt es rund um eine Blockchain, so auch bei Ethereum, immer einen kleinen Kern an Entwicklern, die mit ihrer Arbeit über die Zukunft des Netzwerks entscheiden können, da ihre Entscheidungen nicht auf dem in der Blockchain üblichen Konsens beruhen muss. Somit nehmen diese eine Schlüsselrolle ein. Folgen hieraus sind besonders im Bitcoin-Netzwerk gut zu beobachten. Als die Entscheidung Anstand die Blockgröße zu erhöhen, um mehr Transaktionen in einem Block unterbringen zu können und so den Durchsatz im Netzwerk zu erhöhen, konnte innerhalb der Gemeinschaft kein Konsens gefunden werden. Die Folge war, dass sich mit Bitcoin Cash eine neue Währung von Bitcoin abgespalten hat [15].

Eine *Decentralized autonomous organization (DAO)*, ist ein Versuch dieses Problem in den Griff zu bekommen. Hierbei handelt es sich um einen Algorithmus, der objektiv Entscheidungen für das Netzwerk treffen soll. Gesteuert wird dieser Algorithmus von der Menge aller Mitglieder selbst, die mithilfe von Vorschlägen und Abstimmungen zur Optimierung des Algorithmus beitragen können. Möchte man an einer DAO teilnehmen, kann man sich in diese einkaufen (im Falle von Ethereum mit Ether) und bekommt im Gegenzug anteilige Stimmrechte, wie das gesammelte Geld investiert werden soll. Sollte die DAO zu groß werden oder sich eine zu dominante Gruppe in der DAO bilden, steht

es jedem Teilnehmer jederzeit frei die DAO zu verlassen oder sich von dieser abzuspalten und seine eigene DAO zu gründen. Das Ganze geschieht dabei vollkommen kostenlos und autonom [15].

Anfang 2016 wurde das Konzept einer DAO zum ersten Mal praktisch angewendet und für das Ethereum-Netzwerk umgesetzt. Mithilfe der DAO wollte sich das Ethereum-Netzwerk weiter finanzieren. Für die DAO wollten sich die Entwickler von Ethereum die Vorteile ihres eigenen Netzwerks zu nutzen machen und diese in Form eines Smart Contracts entwickeln. Wer investieren wollte, musste einfach nur Ether an den Smart Contract überweisen und hat dafür dann im Gegenzug unter anderem Stimmrechte über den weiteren Kurs des Netzwerks erhalten. Niemand hatte die Macht das eingezahlte Geld eigenständig zu verwalten, der Algorithmus des Vertrags hatte die vollkommene Kontrolle. In der Ethereum Gemeinschaft heißt es daher auch, der Code ist das Gesetz. Die Maßnahme war zunächst sehr erfolgreich, ca. 15% des gesamten Ether im Netzwerk flossen in die DAO, was damals einem Marktwert von ca. 150 Millionen Doller entsprach [15].

Nach einer Weile stellte sich aber heraus, dass es eine Schwachstelle im Smart Contract für die DAO gab, über welche Ether entwendet wurde. Hierbei wurde von der erwähnten Möglichkeit Gebrauch gemacht sich von der DAO abzuspalten und somit auch sein investiertes Ether mitzunehmen. Ein Fehler im Code hatte unter bestimmten Voraussetzungen dazu geführt, dass das Ether in der ursprünglichen DAO dabei nicht entfernt wurde. Somit konnten mittels *Replay-Attacke* beliebig viele DAO Abspaltungen erzeugt werden, die sich jeweils wieder Ether aus der DAO abzweigten. Als der Fehler entdeckt wurde, konnte zunächst niemand eingreifen, da der Code sinngemäß die höchste Instanz war und eine Hintertür nicht vorgesehen war. Nach einer Welle von Gegenmaßnahmen konnte das Leck aber abgedichtet werden. Zu diesem Zeitpunkt wurde aber bereits Ether im Wert von 50 Millionen Doller entwendet, was einem Drittel der gesamten Einlage entsprach. Aufgrund der Beschaffenheit einer Blockchain, war es natürlich nicht möglich den oder die Schuldigen zu ermitteln [15].

Nun musste eine Entscheidung getroffen werden. *Vitalik Buterin*, der Erfinder von Ethereum, dessen Wort ein großes Gewicht in der Ethereum Gemeinschaft besitzt, hat als Reaktion auf diesen Diebstahl einen *Hard-Fork* vorgeschlagen, der das Netzwerk auf den Stand zurücksetzen sollte, bevor die DAO erstellt wurde. Die Existenz der DAO und all ihre Folgen würde also mit einem Schlag ausgelöscht werden. Dies würde zwar auf der einen Seite alle Einlagen wieder zurückbringen und Streitigkeiten vermeiden, wie die Frage wem das verbliebene Ether gehört, auf der anderen Seite wäre es aber auch ein massiver Bruch mit dem Kerngedanken einer Blockchain. Was einmal in der Blockchain steht, kann nie wieder entfernt werden. So gab es auch kritische Töne, die vorschlugen, mit dem Verlust zu leben und das Netzwerk zu härten [15].

Letztlich wurde ein modifizierter Client erstellt, welche die DAO aus der Geschichte des Netzwerks löschen sollte. Nun hatte also die Stunde der Wahrheit geschlagen. Der Fork konnte nur erfolgreich sein, wenn eine Mehrheit im Netzwerk den modifizierten Client installiert. Sollte sich das Update nicht durchsetzen und nur von einer Minderheit genutzt werden, würden die Konsenzregeln im Netzwerk die neuen Clients einfach nicht als valide akzeptieren und aus dem Netzwerk ausschließen. Da von dem Diebstahl aber ein großer Teil der Gemeinschaft betroffen war, hat sich das Update letztlich durchgesetzt. Fast 99% des gesamten Netzwerks installierten den modifizierten Client und die DAO war damit fast Geschichte. Die verbliebenen 1% der Clients, die das Update nicht installieren wollten, haben nicht mehr den Konsenzregeln der neuen Clients entsprochen und wurden aus dem Netzwerk ausgeschlossen. Anders herum haben die verbliebenen Clients natürlich auch die neuen Clients aus ihrem Netzwerk ausgeschlossen, da diese nicht mehr den ursprünglichen Konsentzregeln genügten. Diese verblieben Clients führen bis heute die Ursprüngliche Ethereum-Blockchain fort. Da es sich im Vergleich zu dem neu entstanden Fork um eine klare Minderheit handelt, konnten diese den Namen Ethereum nicht für sich beanspruchen. Seit dem ist die Ursprüngliche Ethereum-Blockchain unter dem Namen Ethereum Classic bekannt [15].

Der Wert von Ethereum Classic ist im Vergleich zu Ethereum natürlich deutlich geringer. Nichtsdestotrotz sind die Diebe in der Blockchain von Ethereum Classic weiterhin im Besitz ihrer Beute. Es gibt Vermutungen, dass ein Grund für die Existenz von Ethereum Classic jener ist, dass die Diebe eines Tages versuchen könnten, das erbeutete Ether an einer Börse in Doller umzutauschen. An dieser Stelle könnte die Identität der Diebe dann eventuell aufgedeckt werden. Ein weitere Vorteil von Ethereum Classic ist es, dass durch den geringeren Wert des Ethers in dieser Version des Netzwerks, Smart Contracts für kleinere Anwendungen günstiger erstellt werden können und generell Test mit Smart Contracts günstiger sind [15].

## IX. Ausblick

Das Marktforschungsunternehmen *Gartner* veröffentlicht jedes Jahr ihren sogenannten *Hype-Cycle*, in welchem aktuelle Technologien auf ihre Marktreife hin untersucht werden. Nach diesem durchläuft jede neue Technologie im Laufe der Zeit fünf Phasen, von ihrer Entstehung, bis hin zum Massenmarkt. Nach einem anfänglichen Hype um eine Technologie, in dem erste Erfolge verbucht werden, versinkt eine Technologie meistens wieder in der Vergessenheit, bis sie dann in der letzten Phase den Massenmarkt erreicht. Die Blockchain-Technologie ist aktuell in aller Munde und wird nicht nur in Fachkreisen diskutiert, sondern auch in den Massenmedien. Dies deckt sich mit der Einschätzung von Gartner, welche die Blockchain-

Technologie aktuell auch auf ihrem anfänglichen Hype sieht (siehe Abbildung 3) [16].

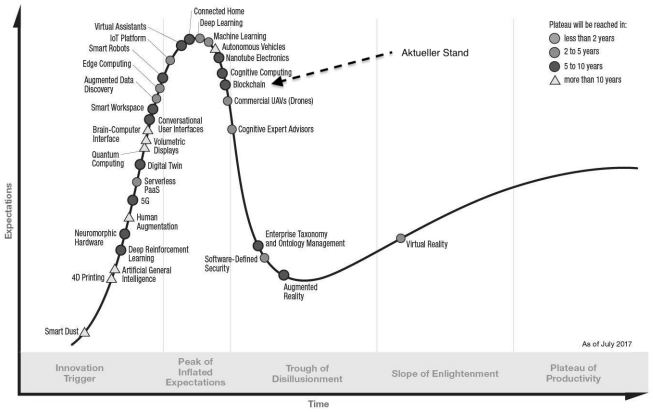


Abbildung 3. Hype-Cycle [16]

Die gesamte Blockchain-Technologie, hat das Potenzial, das digitale Leben nachhaltig zu verändern. Es gibt Stimmen, die sage, die Blockchain ist ein ähnlich großer Meilenstein wie das World Wide Web seiner Zeit. Angefangen hat alles mit dem Bitcoin, aber mittlerweile gibt es Dutzende verschiedene Blockchain Alternativen, die um die Gunst der Zukunft ringen. Ethereum ist nur eine unter diesen, aber gerade auf technischer Seite eine der ausgereiftesten auf dem Markt. Ethereum war die erste Blockchain, die es ermöglicht hat, Smart Contracts auszuführen und hat somit einen Vorsprung auf diesem Feld. Auch besitzt Ethereum eine sehr aktive Community, die sich sehr einig über die Vision ihrer Blockchain ist. Bei Bitcoin sieht dies zum Beispiel ganz anders aus. Es gibt aber auch Probleme, die gelöst werden müssen, möchte Ethereum auch in der Zukunft eine Rolle spielen. Gerade beim *Internet of Things* gibt es andere Blockchains wie IOTA, die sehr viel besser mit den Anforderungen in diesem Bereich zurechtkommen. An einer *C++* Implementierung für Ethereum wird zwar schon gearbeitet, aber wie diese im Vergleich zu anderen abschneiden wird, ist noch unklar. Auch könnte die Finanzierung zum Problem werden. Die Etherum Foundation finanziert sich zurzeit aus den Ether Bestände, die sie bei der Erstellung der Blockchain zurückgehalten hat. Diese Vorräte werden natürlich nicht ewig halten. Für die Blockchain muss in der Zukunft eine solide Finanzierung gefunden werden, damit die Entwickler auch dann noch weiterarbeiten können, wenn die Reserven aufgebraucht sind. Ohne finanziellen Spielraum dürfte es das Netzwerk schwer haben, sich gegen die immer stärkere Konkurrenz durchzusetzen. Auch ein zweites Desaster wie das der DAO könnte dem Netzwerk zum Verhängnis werden [7].

Ob sich Ethereum nun durchsetzt oder nicht, die Blockchain-Technologie wird viele Bereiche des Lebens verändern. Mithilfe von Kryptowährungen können Transaktionen blitzschnell von einem Teil der Erde an den anderen gesendet werden. Wofür das klassische Bankwesen Tage braucht, benötigen Blockchains nur Sekunden. Aber der Finanzsektor ist nur der erste Bereich, in dem sich die Blockchain-Technologie ausbreiten wird. Besitzrechte können in der Blockchain permanent, sicher und kostengünstig gespeichert werden. Smart Contracts werden helfen Verträge deterministisch und durchschaubarer zu machen, sodass viele Rechtsstreitigkeiten vermieden werden können. Auch wird es keine Mittelsmänner mehr geben, was Vertrage günstiger machen wird. Bis dies alles marktreif ist, wird allerdings noch einiges an Zeit vergehen. Vorher gibt es auch noch einige Probleme zu lösen. Die Stärke eines Smart Contracts liegt beispielsweise darin, dass dieser nicht verändert werden kann. Wie sieht es aber aus, wenn dieser aktualisiert werden soll? Hierfür gibt es zurzeit nur unzureichende Workarounds. Dieses und weitere Probleme müssen vorher noch in weiterführenden Arbeiten gelöst werden [7].

## LITERATUR


[1] KAUCHER, ALEXANDER: *Blockchain: Funktionsweise und Applikationsmöglichkeiten*. Hochschule Bonn-Rhein-Sieg, Januar 2018.

[2] KATTWINKEL, OLIVER: *Kryptowährungen*. Hochschule Bonn-Rhein-Sieg, Januar 2018.

[3] NAKAMOTO, SATOSHI: *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf, Oktober 2018.

[4] WATTENHOFER, ROGER: *The Science of the Blockchain*. Inverted Forest Publishing, Erscheinungsort nicht ermittelbar, 1 Auflage, 2016. OCLC: 952079386.

[5] BERENTSEN, ALEKSANDER und FABIAN SCHÄR: *Bitcoin, Blockchain und Kryptoassets*. BoD - Books on Demand, Norderstedt, DE, 1 Auflage, 2017. OCLC: 973719522.

[6] DANNEN, CHRIS: *Introducing Ethereum and Solidity*. Springer Science+Business Media, New York, NY, 1 Auflage, 2017.

[7] DIEDRICH, HENNING: *Ethereum: blockchains, digital assets, smart contracts, decentralized autonomous organizations*. Wildfire Publishing, Lexington, KY, Preview 3 Auflage, 2016. OCLC: 968265659.

[8] ETHEREUM FOUNDATION: *EVM*. http://www.ethdocs.org/en/latest/introduction/what-is-ethereum.html#ethereum-virtual-machine, November 2017.

[9] ETHEREUM FOUNDATION: *Ether*. http://www.ethdocs.org/en/latest/ether.html, Dezember 2017.

[10] ETHEREUM FOUNDATION: *Ethereum Wallet*. https://github.com/ethereum/mist/releases, Dezember 2017.

[11] SZABO, NICK: *The Idea of Smart Contracts*. http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTw Dezember 2017.

[12] GOOS, GERHARD, JURIS HARTMANIS und JAN VAN LEEUWEN: *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Nummer 10436 in *Lecture notes in computer science*. Springer, Cham, 1 Auflage, 2017.

[13] ETHEREUM FOUNDATION: *ERC-20*. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md, Dezember 2017.

[14] ETHEREUM FOUNDATION: *Introduction to Smart Contracts*. https://solidity.readthedocs.io/en/develop/introduction-to-smart-contracts.html, Dezember 2017.

[15] GIESE, PHILIPP, MARK PREUSS, MAXIMILIAN KOPS, SVEN WAGENKNECHT und DANNY DE BOER: *Die Blockchain Bibel: DNA einer revolutionären Technologie*. BTC-Echo, Kleve, DE, 1 Auflage, 2016. OCLC: 962730143.

[16] GARTNER: *Gartner Hype Cycle*. https://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cycle-for-emerging-technologies-2017/, Dezember 2017.