Informatik
Hauptcampus

H O C H
S C H U L E
T R I E R

Ethereum - Mehr als nur Kryptowährung

Bearbeiter: Philip Densborn

Dokumentation zum Fachseminar im Bereich Blockchain

Betreuer: Prof. Dr. Konstantin Knorr

Trier, 27.01.2020

Inhaltsverzeichnis

1 Einleitung 2

2 Funktionsweise 3
2.1 Die Blockchain 3
2.2 Das Mining 5
2.3 Gebühren 6

3 Kryptologische Grundlagen 8
3.1 Ethereum Accounts 8
3.2 Sicherung des globalen Status 8
3.3 Angriffe auf Ethereum 9
3.4 Unterschied ETH und ETC 10

4 Versionen von Ethereum 11
4.1 Ethereum 1.0 11
4.2 Ethereum 2.0 11

5 Smart Contracts 13

6 Praktisches Projekt 19

7 Fazit 21

Literaturverzeichnis 22

Kurzfassung

Das Anlegen von Geld in Kryptowährungen erfreut sich immer größer werdender Beliebtheit. Mit dem Aufschwung von Bitcoin konnte sich Ethereum als zweitgrößte Kryptowährung etablieren. Schon bei der Entwicklung wurde großen Wert darauf gelegt, dass Ethereum nicht nur eine Art von Währung ist, sondern vielmehr eine Plattform für das Erstellen und Ausführen von dezentralisierten Programmen. Diese Programme werden als Smart Contracts bezeichnet und sollen Abhandlungen von Transaktionen automatisiert und ohne Dritte ermöglichen. In realen Anwendungsgebieten könnte ein solcher Vertrag Anwendern beispielsweise den Umweg über eine Bank oder einen Notar ersparen. Der Einsatzbereich ist auf digital erfassbare Ereignisse beschränkt.

1

Einleitung

Seit nun gut zwanzig Jahren sind das Internet und die allgemeine Digitalisierung auf dem Vormarsch. Nahezu alles was digital gespeichert bzw. genutzt werden kann, wird es auch. In Zeiten von Bankenkrisen, Niedrigzinsen und teils hohen Geld-Wechselkursen entstanden neuartige, digitale Wertanlagen bzw. Zahlungsmittel - die Kryptowährungen. Nach der wohl bekanntesten digitalen Währung "Bitcoin" folgten noch zahlreiche weitere dieser Art. Eine davon genannt "Ether". Erfunden wurde dieses System hauptsächlich von Vitalik Buterin, Jeffrey Wilcke und Gavin Wood. Buterin verfasste im Jahr 2013 das sogenannte "Whitepaper", die grundlegende Idee zu Ethereum [Vit13]. Herr Wilcke entwickelte die erste Version mithilfe der Programmiersprache GO [Fai18]. Im Jahr 2014 veröffentlichte Gavin Wood dazu die erste formale Spezifikation von Ethereum und der EVM (Ethereum Virtual Machine) [Gav14]. Der eigentliche Betrieb begann im Juli 2015. Schon von Anfang an wurde Ethereum so designt, dass es sich nicht nur um eine dezentralisierte Währung handelt, sondern mehr um dezentrale Applikationen, die auf der Blockchain ausgeführt werden können.
In dieser Arbeit werden die Prinzipien und Funktionen von Ethereum grundlegend erläutert. Am Ende wird ein kurzes Fazit gezogen.

Abb. 1.1.
Ethereum Logo

2

Funktionsweise

Zu Beginn, um die Funktionsweise von Ethereum überhaupt verstehen zu können, muss die Frage geklärt werden, was "Ethereum" denn eigentlicht ist. Die Meisten verbinden mit Ethereum eine Kryptowährung wie z.B. Bitcoin. Dies ist soweit auch gar nicht falsch, jedoch ist Ethereum weit mehr als nur das. Die zugrundeliegende Basis, für diese offene Plattform ist die Blockchain-Technologie. Darauf können Entwickler dezentralisierte Applikationen entwickeln und ausführen. Der Begriff "Ethereum" beschreibt daher die Technologie des Systems im Ganzen und "Ether" (ETH) die Währung im eigentlichen Sinne [Disb]. Im Nachfolgenden wird dieses System genauer erklärt.

2.1 Die Blockchain

Wie auch Bitcoin basiert Ethereum auf einer Blockchain. Aber was ist denn eine "Blockchain"? Einfach gesagt ist eine Blockchain eine öffentliche, dezentrale Datenbank in der getätigte Transaktionen in Form von aneinander gereihten Blöcken gespeichert werden. Dabei wird es Menschen und Computern ermöglicht, sich auf Dinge zu einigen, ohne sich gegenseitig zu kennen bzw. sich vertrauen zu müssen. Das Besondere an Ethereum ist, dass nicht nur Zahlungen gespeichert werden, sondern vielmehr "Computerprogramme" [Disa]. Diese Programme, genannt "Smart Contracts", werden in einem eigenen Kapitel behandelt.
Jeder Block der Blockchain enthält sogenannte "Metainformationen". Unter anderem sind dies beispielsweise:

- der Blockname,
- der Zeitstempel der Erzeugung des Blocks,
- die Transaktionen des Blocks,
- die Kosten der Erzeugung des Blocks (Gas),
- der/die Erzeuger des Blocks uvm.

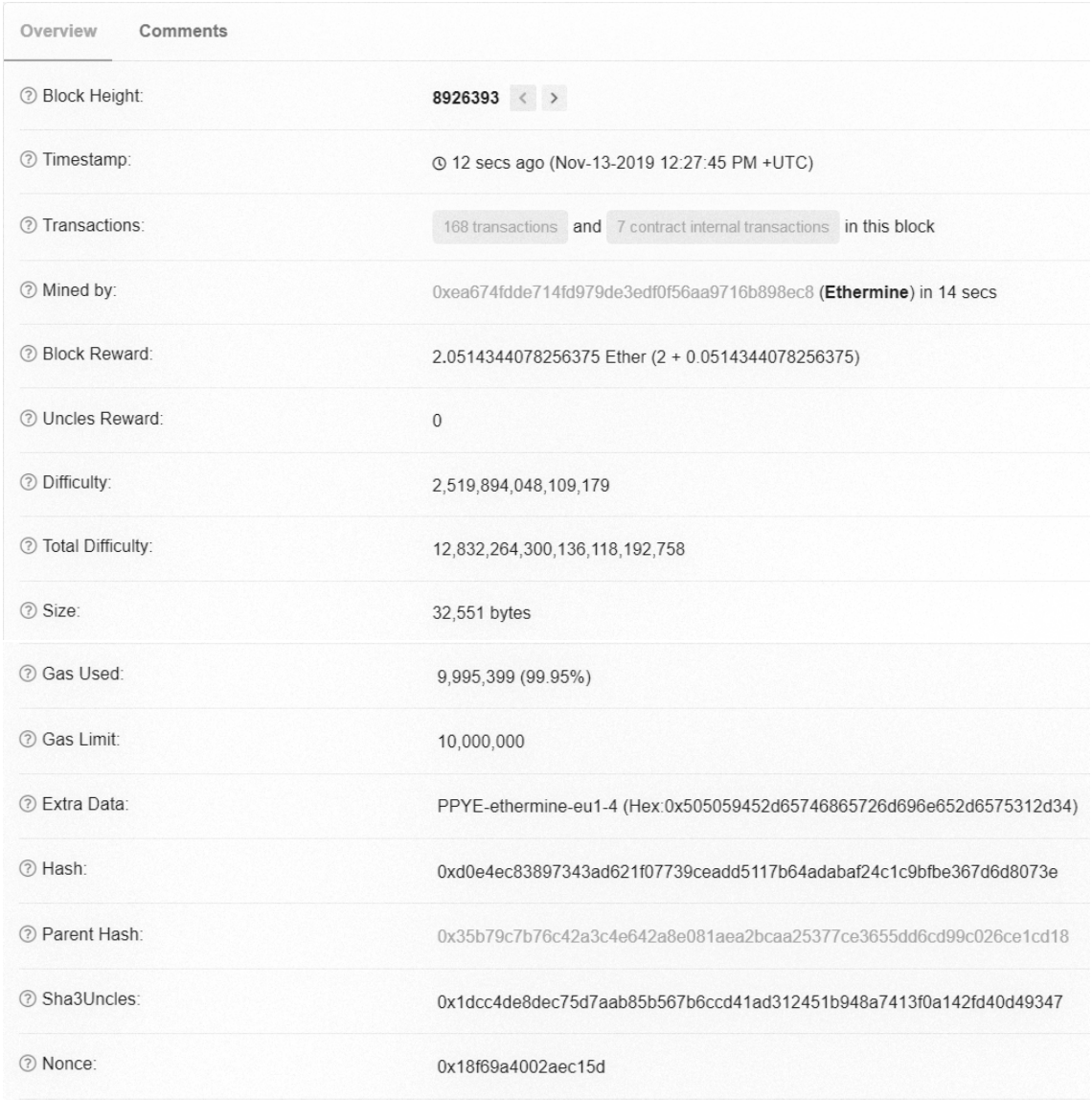

Abb. 2.1. Blockinformationen von Block 8926393. Zufällig ausgewählt auf der Seite "`https://etherscan.io`"

Dadurch, dass jeder Block den Hashwert des vorangegangenen Blocks enthält (siehe Abb. 2.2), entsteht ein zusammenhängendes System. Der einzige Block, der nicht den Hashwert des vorherigen Blocks enthält, ist der sogenannte "Genesis-Block" (der erste Block in der Blockchain).
Das Protokoll von Ethereum ist so ausgelegt, dass ca. alle 14 Sekunden ein neuer Block erzeugt wird (bei Bitcoin ca. alle 10 Minuten). Aufgrund der schnellen Erzeugung sind die Blöcke der Blockchain sehr klein (KB - Bereich) [Ant16]. Die Größe bei Bitcoin beträgt im Durchschnitt 1MB [BT]. In einem Block werden durchschnittlich zwischen 100-300 Ethereum- und 1500-2000 Bitcoin-Transaktionen zusammengefasst. Bei Ethereum kann die Anzahl stark variieren, da sie direkt von den benötigten Gebühren (Gas) des Ethereum-Netzwerkes abhängen [Ant16].

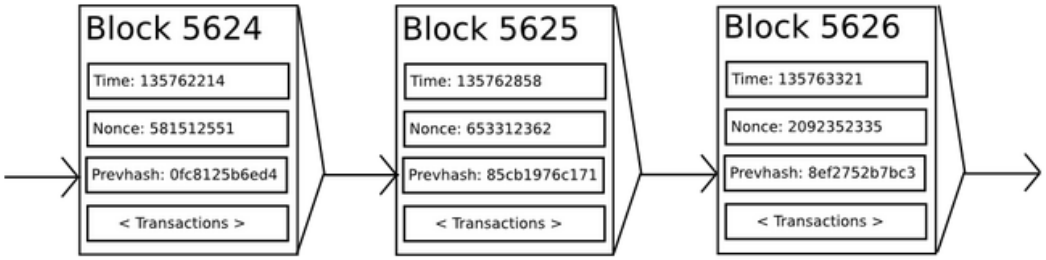

Abb. 2.2. Beispiel einer Verkettung innerhalb einer Blockchain. Abbildung entnommen aus dem Ethereum-Whitepaper [Vit13]

2.2 Das Mining

Ether fällt wie das meiste im Leben nicht einfach vom Himmel. Doch wie entsteht ein Block in der Blockchain? Dies geschieht mit einem mathematischen Beweis. Ein "Miner" muss quasi seinen erstellten Block schneller als alle anderen Miner validieren. Dieses Verfahren wird "Proof-of-work" genannt. Dazu muss ein mathematischer Wert errechnet werden. Dieser wird nur durch ständiges Ausprobieren "gefunden". Der verwendete Algorithmus zum errechnen des Hashwertes wird bei Ethereum "Ethash" genannt. Bitcoin verwendet den SHA-2 Algorithmus. Hat ein Miner einen neuen Block errechnet, schickt er diesen an alle weiteren Nodes (Clients bzw. Miner) des Netzwerkes. Diese überprüfen dann die Richtigkeit des Blocks. Nach dem erfolgreichen Validieren wird der Miner für den entstandenen rechnerischen Aufwand mit einer gewissen Anzahl von Ether entlohnt [Coi19]. Weil die Zeit zwischen der Erzeugung von neuen Blöcken sehr kurz ist, kann es zu Überschneidungen bei der Blockerzeugung kommen. Diejenigen, die es nicht geschafft haben den Block als erstes zu minen, bekommen einen kleineren Betrag, als den sie eigentlich erhalten hätten. Man nennt diese Blöcke "Uncles" [Ant16]. Bei einem echten "Uncle" wären es 7/8 ETH, bei noch einer Ebene darunter nur noch 6/8 ETH usw. Das GHOST-Protokoll ist bis auf eine Tiefe von sieben beschränkt [Vit13]. Der zu spät erzeugte Block wird aber nicht an die Hauptkette "angekettet" sondern nebenan "angedockt" [Ant16]. Das Ziel des Mining ist in erster Linie nicht die Erstellung neuer Währungs-Token, sondern vielmehr die Sicherung der Blockchain [AW18].

GHOST Protokoll

Damit alle Miner zur selben Blockchain beitragen und nicht Ihre eigenen "Wege" gehen, wurde das sogenannte "GHOST Protokoll" (Greedy Heaviest Observed Subtree) erfunden. Dieses besagt, dass im Falle eines "Forks" (einer Abtrennung der Blockchain), immer der Kette vertraut werden soll, welche die am meisten aufgewendete Rechenkraft enthält (siehe Abb. 2.3). Überprüfen lässt sich dies z.B. anhand der Blocknummer des zuletzt erstellten Blocks oder der insgesamten "Difficulty" (Schwierigkeit der Blockchain) [Pre17]. Bei dieser Art von Vorgehen kom-

men dem Netzwerk wieder die “Uncles” zugute. Diese tragen unmittelbar zur Sicherheit des Netzwerkes bei. Jedere Miner der einen “Uncle” inkludiert, bekommt 3.2% des aktuellen ETH-Rewards extra als Gewinn hinzu.

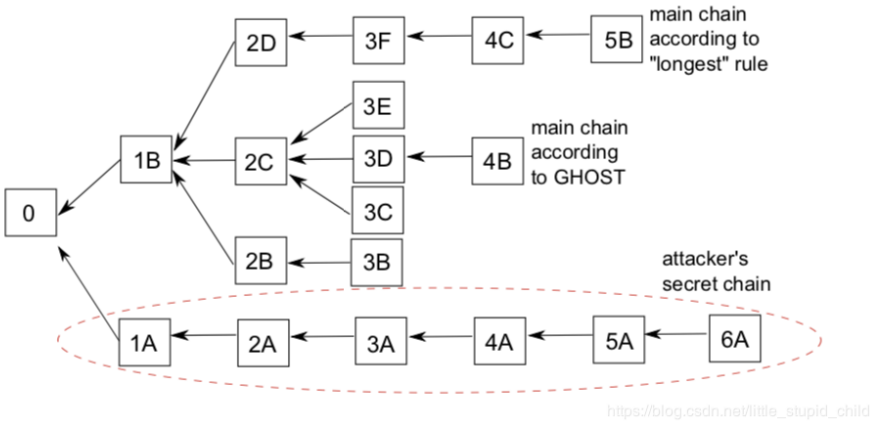

Abb. 2.3. Beispiel des GHOST-Protokolls. Abbildung entnommen von https://appserversrc.8btc.com

2.3 Gebühren

Wie bei allen angebotenen Diensten, fallen auch bei Ethereum Kosten an. Diese entstehen bei jeder neuen Transaktion für das Herunterladen und Verifizieren der Blockchain. Sie dienen nicht ausschließlich dazu das Netzwerk zu unterhalten, sondern auch um das Netz vor Missbrauch zu schützen [Pre17]. Bei Ethereum werden diese Gebühren “Gas” genannt. Möchte eine Person Transaktionen tätigen, sendet sie den Wunsch auf Ausführung dieser Transaktionen raus an das Ethereum-Netzwerk. Ein Miner wird diese Aktion dann auszuführen. Der Gas-Preis wird dabei von der Person festgelegt die die Transaktion ausführen soll. Je höher der eigens angebotene Gas-Preis ist, desto höher ist die Wahrscheinlichkeit, dass die gewünschte Transaktion zügig ausgeführt wird [Ant16].

Units in Ethereum		
Unit	**Number per ETH**	**Most appropriate uses**
Ether (ETH)	1	Currently used to denominate transaction amounts (eg 20 ETH) and mining rewards (5 ETH)
finney	1,000	
szabo	1,000,000	Currently the best unit for the cost of a basic transaction, eg 500 szabo
Gwei	1,000,000,000	Currently the best unit for Gas Prices eg 22 Gwei
Mwei	1,000,000,000,000	
Kwei	1,000,000,000,000,000	
wei	1,000,000,000,000,000,000	The base indivisible unit used by programmers

Abb. 2.4. Einheiten der Ether-Währung. Abbildung entommen aus [Ant16].

3

Kryptologische Grundlagen

In diesem Kapitel werden die kryptologischen Grundlagen hinter Ethereum beschrieben. Im Kern basiert das System auf symmetrischen und asymmetrischen Verschlüsselungsverfahren. Das am meisten genutzte Hashverfahren ist Keccak256. Auch wenn SHA3 darauf aufbaut, handelt es sich nicht um das standardisierte SHA3 sondern eben um das Ursprungsverfahren (da durch Edward Snowden herauskam, dass sich die NSA Backdoors in das Verfahren von SHA3 einbauen lassen wollte) [AW18].

3.1 Ethereum Accounts

Ein Ethereum Account wird durch eine 20 byte bzw. 160 bit hexadezimale Adresse identifiziert (siehe Reiter "Hash" Abb. 2.1). Dies sind die letzten 20 byte des Public Keys. Im Grunde gibt es zwei Arten von Accounts [Pre17]:

1. Externe-, und
2. Kontrakt-Accounts (Smart Contracts)

Externe Accounts können Transaktionen zu anderen Externen-, als auch Kontrakt-Accounts senden, indem sie die Nachricht mit ihrem privaten Schlüssel signieren. Die Kontrakt-Accounts können dies ohne externes Einwirken nicht.
Jeder Externe-Account hat folgende Bestandteile [Pre17]:

- eine Nonce: Sie gibt die Anzahl an getätigten Transaktionen wieder,
- den aktuellen Ether Stand in der Einheit Wei,
- der Speicher eines Accounts, der bei Externen-Accounts leer ist,
- und den "CodeHash" der ebenfalls, bei Externen-Accounts, leer ist.

3.2 Sicherung des globalen Status

Damit die Integrität und Vertraulichkeit der Blockchain gewährleistet ist, verwendet Ethereum zum Abspeichern einen "Merkle Patrica Tree". Eine solche Darstellung, die immer zwei "Kinder" enthält, wird "binärer Baum" genannt (siehe Abb. 3.1).

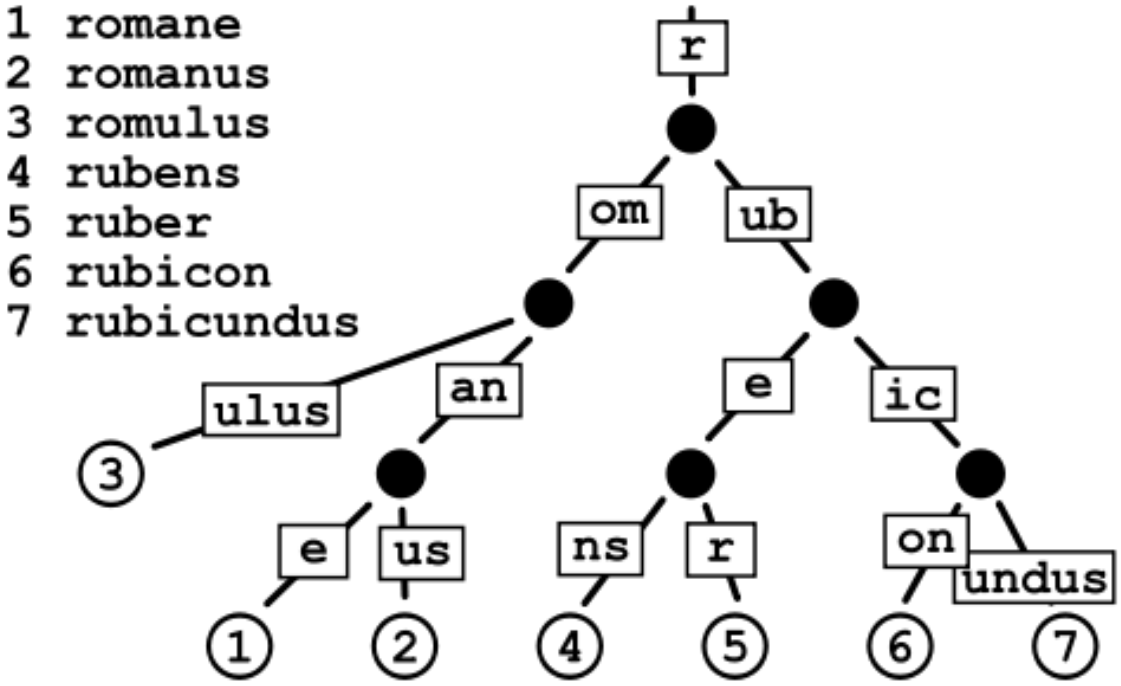

Abb. 3.1. Abbildung zeigt einen Merkle-Patricia-Baum

Bei Ethereum handelt es sich aber nicht nur um einen einfachen binären Baum sondern um drei Bäume. Diese sichern jeweils [Vit15]:

1. die Transaktionen,
2. die Auswirkungen bzw. Effekte der Transaktionen und
3. den Status ab.

Gerade diese Art der Speicherung ermöglicht es Light-Clients verschiedene Anfragen einfach zu tätigen. Zur Verifizierung ob ein Block an der richtigen Stelle steht, können die einzelnen Hashes bis hin zum "Root-Hash", der oberste und erste Hash eines Baumes, überprüft werden [Vit15]. Eine detaillierte Beschreibung der "Merkle-Patricia-Trees" findet sich im Ethereum-Yellowpaper [Gav14].

3.3 Angriffe auf Ethereum

Trotz der Nutzung der Blockchain-Technologie ist Ethereum nicht vollkommen sicher. Da es einige, wenn teilweise auch nur theoretische Schwachstellen gibt, wird in diesem Dokument exemplarisch nur auf zwei einzelne eingegangen:

Reentrance-Attacke

Bei diesem Angriff ist es möglich, den Softwarecode von Smart Contracts wiederholt auszuführen. Es gibt verschiedene Variationen dieser Art, eine der am besten ausgenutzten ist jedoch die Reentrance bei einer einfachen Funktion.

```
1     mapping (address => uint) private userBalances;
2
3     function withdrawBalance() public {
4        uint amountToWithdraw = userBalances[msg.sender];
5        // At this point, the caller's code is executed, and can call
6        // withdrawBalance again
7        (bool success, ) = msg.sender.call.value(amountToWithdraw)("");
8        require(success);
9        userBalances[msg.sender] = 0;
10    }
```

Beispiel Funktion eines Smart Contracts. Entnommen aus [Con]

An diesem Beispiel ist gut zu erkennen, dass wenn die Funktion "withdrawBalance()" nochmals aufgerufen wird solange sich der erste Aufruf noch in Zeile sieben befindet, der Kontostand danach inkonsistent ist. Die Auswirkung dieses Fehlers wird in 3.4 genauer beschrieben [Con].

51% Attacke

Dieser Angriff ist praktisch meist nur unter dem Verfahren "Proof-of-Work" möglich. Sollte es vorkommen, dass ein Miner bzw. ein Mining-Pool mehr als 50% der Rechnerpower aufbringt, könnte es vorkommen, dass Ether doppelt ausgegeben werden können. Dafür erstellt der Angreifer eine neue Blockchain mit der selben (oder längeren) Länge der originalen Kette, jedoch einer anderen Transaktions-Historie. Es können dabei aber keine bestehenden Transaktionen oder Regeln geändert werden [Gar19].

3.4 Unterschied ETH und ETC

Auf Kryptobörsen wird oftmals von Ethereum und Ethereum Classic gesprochen. Doch worin unterscheiden sie sich? Für das Verständnis muss zuvor noch geklärt werden, was eine DAO (Dezentralisierte Autonome Organisation) ist. Im Gegensatz zu normalen Organisationen ist eine DAO eine Organisation die ohne Aktionäre oder eine zentrale Instanz auskommt. Im Jahre 2016 gelang es Hackern eine solche Organisation namens "The DAO" zu hacken. Dabei wurden Ether im Wert von ca. 50 Millionen US-Dollar gestohlen [Max16]. Zur Gegenwirkung wurde im Anschluss daran von dem größten Teil der "Ethereum-Gemeinde" und dem Gründer Buterin festgelegt einen sogenannten "Hard-Fork" durchzuführen. Dies bedeutete die eigentliche Blockchain wurde in zwei geteilt und damit der ursprüngliche Zustand wiederhergestellt. Dieser Schritt stieß auf großen Gegenwind, da so in ein eigentlich dezentrales System eingegriffen wurde. Seit diesem Zeitpunkt werden beide Blockchains und Währungen weiter fortgeführt. Die beiden Währung heißen heute Ether(ETH) und Ether Classic (ETC). Dabei ist die offiziell von der Gemeinde und Erfinder weiter unterstütze Währung die Neuere ETH [Osm18].

4

Versionen von Ethereum

Die erste Version von Ethereum wurde im Juli 2015 in Betrieb genommen. Danach folgten weitere Versionen, die Verbesserungen enthielten. Mit Ethereum 2.0 soll das System von "Proof-of-Work" auf "Proof-of-Stake" umgestellt werden.

4.1 Ethereum 1.0

Mit im Vorhinein 11,9 Millionen erstellten Ether ging die erste Version 2015 online. Der Entwicklungsstand von Ethereum lässt sich in vier Phasen einteilen [ET]:

- Frontier
- Homestead
- Metropolis
- Serenity (Ethereum 2.0)

Frontier: Die erste Version von Ethereum war eher für Leute mit dem technischen Now-How gedacht.
Homestead: Wurde am 14.März 2016 eingeführt. Mit diesem Update wurden einige Protokollverbesserungen, insbesondere die schnellere Handhabung von Transaktionen und IT-Sicherheitsaspekten, vorgenommen. Von diesem Zeitpunkt an galt Ethereum als "sicher" zu nutzen.
Metropolis: Die aktuelle Phase seit 2017. Von hier an ist es möglich "Light-Clients"[1] zu nutzen und später, ca. ab dem Jahr 2020 soll ein DApp-Store (Dezentralisierter Applikations-Store) eingebunden werden.

4.2 Ethereum 2.0

Der große Unterschied zwischen Ethereum 1.0 und 2.0 liegt im Block-Erstellungs-Konsensus von "Proof-of-Work" zu "Proof-of-Stake". Statt einen Block über ein kryptografisches Puzzle zu validieren, wird nun über eine Art Gewichtung validiert. Das heißt, dass eine Abstimmung über den nächsten zu ergänzenden Block

[1] Ein Light-Client dient dazu, Transaktionen in der Ethereum-Blockchain zu validieren. Normalerweise muss dafür immer die gesamte Blockchain heruntergeladen werden. Dies entfällt bei diesen und so wird nur ein gewisser Teil geladen, um den PC des Nutzers zu entlasten.

abgehalten wird. Jeder der daran teilnehmen möchte, muss einen bestimmten Teil Ether (zur Zeit sind mind. 32 ETH geplant) als Sicherheitsaufbewahrung in ein Deposit einzahlen. Für diese Einzahlung wird im Hintergrund der sogenannte "Casper-Contract" ausgeführt, benannt nach dem verwendeten Protokoll namens "Casper". Die Gewichtung der Stimme hängt dann von den im Deposit eingesperrten Ethern ab. Für die Erzeugung eines neuen Blocks wird ein Validierer per Pseudo-Zufallsverfahren ausgelost (Im Proof-of-Stake nennt man diese "Miner" dann "Schmiede"). Bei der zufälligen Auswahl wird der nächste Validierer anhand der niedrigsten Hashrate und dem größten Stake ausgewählt. Da alle Stakes (Deposits) öffentlich einsehbar sind, können alle Nodes die nächste Wahl vorhersagen. Sobald ein neuer Block erzeugt wurde, kontrollieren bzw. validieren die Nodes des Netzwerkes diesen Block und "stimmen" für ihn ab. Sollte ein Schmied bei der Erzeugung eines Blocks Fehler machen bzw. versucht zu betrügen, verliert er all seine Coins im Deposit. Macht der Erzeuger alles richtig, bekommt er einen Reward und die zum Block gehörigen Transaktionsgebühren. Wenn die eingelagerten und damit gesperrten Ether entnommen werden sollen, muss erst mindestens 18 Stunden gewartet werden. Es kann ggf. auch länger dauern, je nachdem wie viele Personen ihr Ether zur Zeit "abheben" möchten. Dieser Ansatz ist im Gegensatz zu "Proof-of-Work" deutlich Strom sparender, im Allgemeinen sicherer und eliminiert das Risiko einer Zentralisierung [Eth].

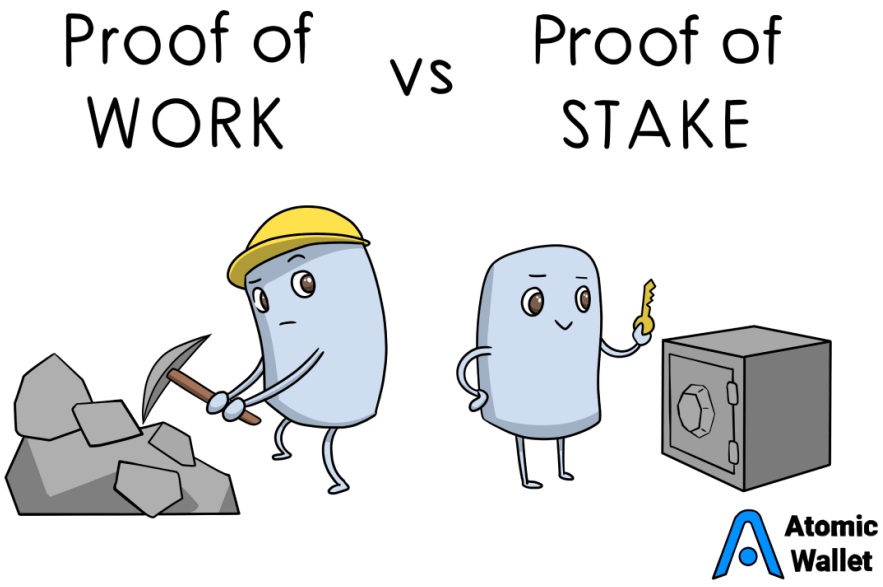

Abb. 4.1. Proof-of-Work vs. Proof-of-Stake.

5

Smart Contracts

Wie schon in der Einleitung zum Kapitel Funktionsprinzipien erläutert, sind Smart Contracts Programme, die auf der Ethereum-Blockchain laufen. Doch wie sehen diese aus und wie läuft das ganze Prozedere ab? Genau diese Fragen werden im folgenden Kapitel aufgeschlüsselt.

Was sind Smart Contracts?

Smart Contracts bestehen im Prinzip aus Softwarecode. Dieser ist auf der Blockchain aufgesetzt und führt automatisch Transaktionen zwischen mehreren Teilnehmern aus. Zuerst wird dafür die gewünschte Funktion eines solchen "Vertrages" in einer höheren Programmiersprache geschrieben. Anschließend muss dieser in eine, für die Blockchain, verständliche Sprache compiliert werden [WF19]. Für Ethereum wäre das EVM-Bytecode (Ethereum Virtual Machine-Bytecode) [AW18].

Wozu dienen Smart Contracts?

Es gibt einige gute Gründe, die den Einsatz von Smart Contracts rechtfertigen [WF19].

Keine dritten Personen zwischen den Transaktionen

Durch das automatisierte Abhandeln der Transaktionen ist es möglich Aufträge auszuführen, ohne dass sich die unterschiedlichen Parteien gegenseitig kennen bzw. vertrauen müssen. Dies bedeutet, dass keine Anwälte oder Banken als Verhandlungsleiter bzw. -partner benötigt werden.

Transparenz und Rechtssicherheit

Dadurch, dass jede getätigte Transaktion dauerhaft in der Blockchain festgehalten wird, können Einträge nicht nachträglich manipuliert werden. Da alle Transaktionen nachverfolgt werden können, bietet sich eine hohe Transparenz. Gegenüber den Nutzern entsteht so auch eine Rechtssicherheit. Jedoch muss natürlich die richtige und sachgemäße Programmierung des Smart Contractes gewährleistet sein.

Anonymität

Generell ist das Nutzen von Smart Contracts anonym. Der Public Key des Contracts wird im Gegensatz zu einem externen Account im Netzwerk der Blockchain gespeichert, d.h. Nutzer kommunizieren direkt mit ihm und nicht über Umwege. Es werden nur anonyme Transaktionsdaten bei Nutzung des Vertrags wie z.B. der Public Key, das Datum und der Betrag gespeichert.

Als Anmerkung ist noch festzuhalten, dass Smart Contracts beschränkt auf digital überprüfbare Ereignisse sind. Dies bedeutet sie treffen einfache true/false Aussagen. Alles was darüber hinaus geht, kann nicht von Smart Contracts abgehandelt werden.

Gas-Gebühren bei Smart Contracts

Bei der Erstellung bzw. Nutzung eines Smart Contracts entstehen Kosten. Diese werden anhand der verwendeten Operationen innerhalb des Vertrages berechnet. Die dabei genutzten "Low-Level" Befehle werden "Opcodes" genannt und sind im Ethereum-Yellowpaper [Gav14] unter dem Kapitel "Appendix G. Fee Schedule" genaustens beschrieben. Der eigentliche Preis wird dann mit der Formel: GAS_AMOUNT * GAS_PREIS berechnet.

G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.

W_{zero} = {STOP, RETURN, REVERT}
W_{base} = {ADDRESS, ORIGIN, CALLER, CALLVALUE, CALLDATASIZE, CODESIZE, GASPRICE, COINBASE, TIMESTAMP, NUMBER, DIFFICULTY, GASLIMIT, RETURNDATASIZE, POP, PC, MSIZE, GAS}
$W_{verylow}$ = {ADD, SUB, NOT, LT, GT, SLT, SGT, EQ, ISZERO, AND, OR, XOR, BYTE, CALLDATALOAD, MLOAD, MSTORE, MSTORE8, PUSH*, DUP*, SWAP*}
W_{low} = {MUL, DIV, SDIV, MOD, SMOD, SIGNEXTEND}
W_{mid} = {ADDMOD, MULMOD, JUMP}
W_{high} = {JUMPI}
$W_{extcode}$ = {EXTCODESIZE}

Abb. 5.1. Ausschnitte aus dem Etherem-Yellowpaper zur Smart Contract Gas-Gebühren-Berechnung.

Wie sieht ein Smart Contract aus?

Ein Smart Contract wird bei Ethereum meist in der Sprache "Solidity" geschrieben (es gibt aber auch andere Sprachen). Die Syntax dieser Programmiersprache ähnelt der von JavaScript, Python und C++. Die Abbildungen 5.2 und 5.3 zeigen einen Ausschnitt aus dem praktischen Projekt, welches nach dem Vortrag zu dieser Ausarbeitung stattfinden wird. Das Attribut "address" speichert einen 160 bit Wert

auf dem keine weiteren mathematischen Operationen ausgeführt werden können. Die Zuweisung von “mapping (address => uint) public balances” arbeitet wie eine Art Hashmap. Eine Adresse wird hier in einen Integerwert zugeordnet. Mit “event FundingReceived(address contributor, uint value, uint currentTotal)” ist es einem Client beispielsweise möglich auf eingehende Spenden zu reagieren. Dieses wird im Beispiel in der sechsten Zeile der “contribute()-Methode” gesendet. Eine weitere besondere Syntax ist hier z.B. “msg.sender”. Mit diesem Aufruf wird die Adresse des Aufrufers des Vertrages zurückgegeben [Sol]. Das Signalwort “paypable” lässt es zu, dass auf eine Adresse oder einen Vertrag, Währung ein- bzw. ausgezahlt werden kann. Eine Funktion die mit “external” gekennzeichnet wurde, kann nur außerhalb eines Vertrages aufgerufen werden. Logischerweise kann eine Funktion mit dem Zusatz “internal”, dann nur von innerhalb eines Vertrages aufgerufen werden. “Modifier” wie z.B. “modifier inState(State state)” dienen dazu, Funktionen um gewisse Bedingungen zu erweitern. Die Funktion “require()” wirkt wie eine Art if-Statement. Ist die Bedingung nicht erfüllt wird ein Fehler geworfen [AW18].

```
contract Project {
    using SafeMath for uint256;

    // Data structures
    enum State {
        Fundraising,
        Expired,
        Successful
    }

    // State variables
    address payable public creator;
    uint public amountGoal; // required to reach at least this much, else everyone gets refund
    uint public completeAt;
    uint256 public currentBalance;
    uint public raiseBy;
    string public title;
    string public description;
    State public state = State.Fundraising; // initialize on create
    mapping (address => uint) public contributions;

    // Event that will be emitted whenever funding will be received
    event FundingReceived(address contributor, uint value, uint currentTotal);
    // Event that will be emitted whenever the project starter has received the funds
    event CreatorPaid(address recipient);
    // Event that will be emitted whenever the project has been expired
    event ProjectExpired(bool expired);

    // Modifier to check current state
    modifier inState(State _state) {
        require(state == _state, "Not funding anymore");
        _;
    }

    // Modifier to check if the function caller is the project creator
    modifier isCreator() {
        require(msg.sender == creator, "Sender is not creator!");
        _;
    }
```

Abb. 5.2. Solidity Code Beispiel

```
constructor
(
    address payable projectStarter,
    string memory projectTitle,
    string memory projectDesc,
    uint fundRaisingDeadline,
    uint goalAmount
) public {
    creator = projectStarter;
    title = projectTitle;
    description = projectDesc;
    amountGoal = goalAmount;
    raiseBy = fundRaisingDeadline;
    currentBalance = 0;
}

/** @dev Function to fund a certain project.
  */
function contribute() external inState(State.Fundraising) payable {
    require(msg.sender != creator, "Sender is creator!");
    bool expired = isExpired();
    contributions[msg.sender] = contributions[msg.sender].add(msg.value);
    currentBalance = currentBalance.add(msg.value);
    emit ProjectExpired(expired);
    emit FundingReceived(msg.sender, msg.value, currentBalance);
    if(!expired) {
        checkIfFundingComplete();
    }
}

/** @dev Function to change the project state depending on conditions.
  */
function checkIfFundingComplete() internal {
    if (currentBalance >= amountGoal) {
        state = State.Successful;
        payOut();
    }
    completeAt = now;
}
```

Abb. 5.3. Fortsetzung Solidity Code Beispiel

Wie kann eine Anwendung auf einen Smart Contract zugreifen?

In dem hier demonstrativen Beispiel (siehe Abb. 5.4) wurde eine Web-App mit "Web3.js" programmiert. Zuerst werden für die Interaktion mit den Smart Contracts die benötigten Dateien importiert. In diesen ist die Grundstruktur der verwendeten Verträge festgehalten. Diese Struktur wird als "Application Binary Interface", kurz ABI, bezeichnet. So kann ein Programm den Maschinen-Code des Vertrages richtig interpretieren.

```
<script>
import crowdfundInstance from '../contracts/crowdfundInstance';
import crowdfundProject from '../contracts/crowdfundProjectInstance';
import web3 from '../contracts/web3';

export default {
  name: 'App',
  data() {
    return {
      startProjectDialog: false,
      account: null,
      stateMap: [
        { color: 'primary', text: 'Ongoing' },
        { color: 'warning', text: 'Expired' },
        { color: 'success', text: 'Completed' },
      ],
      projectData: [],
      newProject: { isLoading: false },
    };
  },
  mounted() {
    // this code snippet takes the account (wallet) that is currently active
    web3.eth.getAccounts().then((accounts) => {
      [this.account] = accounts;
      this.getProjects();
    });
  },
  methods: {
    fundProject(index) {
      if (!this.projectData[index].fundAmount) {
        return;
      }

      const projectContract = this.projectData[index].contract;
      this.projectData[index].isLoading = true;
      projectContract.methods.contribute().send({
        from: this.account,
        value: web3.utils.toWei(this.projectData[index].fundAmount, 'ether'),
      }).then((res) => {
        const expired = res.events.ProjectExpired.returnValues.expired;
        const newTotal = parseInt(res.events.FundingReceived.returnValues.currentTotal, 10);
        const projectGoal = parseInt(this.projectData[index].goalAmount, 10);
        this.projectData[index].currentAmount = newTotal;
        this.projectData[index].isLoading = false;

        // Set project state to expired
        if(expired == true) {
          this.projectData[index].currentState = 1;
        }
        // Set project state to success
        else if (newTotal >= projectGoal) {
          this.projectData[index].currentState = 2;
        }
      });
    },
```

Abb. 5.4. Web3.js-Smart Contract Implementierungsbeispiel

In Zeile 223 ist eine Methode namens “fundProject(index)” definiert. Dort wird in Zeile 228 auf das passende Projekt zugegriffen. Die Methode “contribute()” des Vertrages wird anschließend in Zeile 230 aufgerufen. Von Zeile 234 bis Zeile 247 kann auf das Ergebnis der Ausführung reagiert werden. Bei den Zeilen 234 und 235 wird beispielsweise auf das Ergebnis von zwei verschiedenen Events reagiert, die von einem Smart Contract gesendet wurden.

6

Praktisches Projekt

Als praktischer Teil dieser Arbeit, werden Smart Contracts anhand eines Crowdfunding-Beispiels demonstriert 6.1. Dafür können sich alle anwesenden Kommilitoninnen und Kommilitonen das Plugin “Metamask” für Ihren Browser herunterladen und einen Wallet erstellen. Über “https://faucet.metamask.io/” kann sich Test-Ether besorgt werden. Der Smart Contract wird im Anschluss daran von mir mithilfe der Remix-IDE auf die Blockchain des Ropsten-Testnetzwerkes geschrieben und erläutert. Danach wird der eigentliche Webserver (externer Server) passend für den Smart Contract justiert. Nach dem richtigen Einstellen des Servers wird er gestartet, und es kann damit begonnen werden Crowdfunding-Projekte zu erstellen bzw. zu unterstützen. Während dessen können alle getätigten Transaktionen auf “etherscan.io” eingesehen werden.

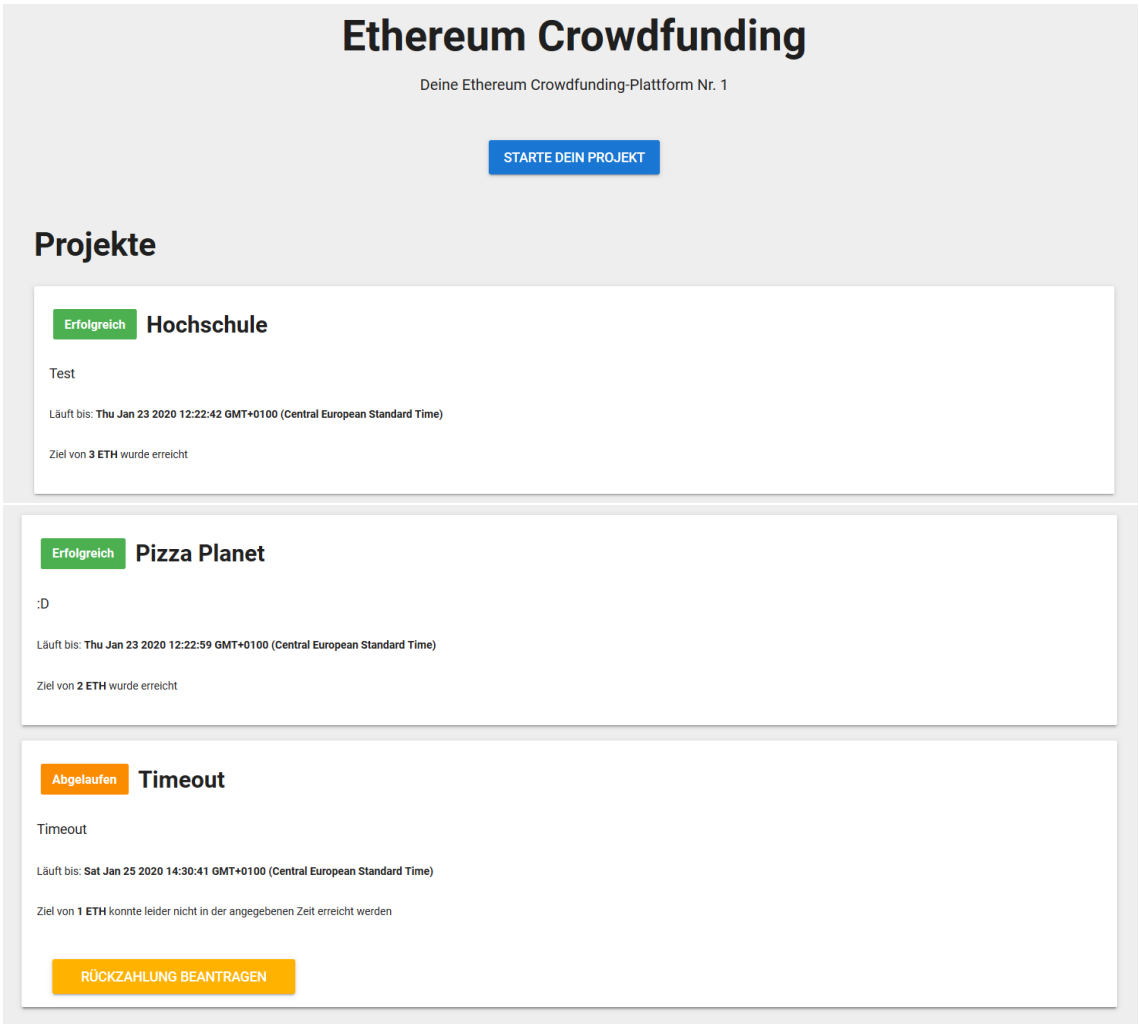

Abb. 6.1. Crowdfunding-Projekt

7

Fazit

Ethereum besitzt eine eigene Währung namens Ether (ETH) und die Möglichkeit, Programme auf der Blockchain auszuführen. Mit Hilfe dieser ist es möglich Abhandlungen aus der realen Welt digital und sicher auszuführen. Jede getätigte Transaktion ist in der Blockchain einsehbar. Die größten Probleme von Ethereum sind wohl der noch aktuell verwendete Mining-Algorithmus "Proof-of-Work" und die allgemein sehr langsame Transaktionsrate von ca. 15 Transaktionen pro Sekunde. Mit der Einführung von Ethereum 2.0 soll auf das Verfahren "Proof-of-Stake" zu Gunsten der Umwelt und Sicherheit umgestellt werden. Dabei soll schlussendlich ebenfalls die Transaktionsrate deutlich gesteigert werden.
Es gibt durchaus kritische Stimmen, denen z.B. die gesamte Roadmap von Ethereum nicht ambitioniert genug ist [Adr19] oder allgemein die Blockchain-Technologie zu sehr "gehyped" wird [Bru19]. Egal ob Befürworter oder Kritiker, eines steht fest: Die nächsten Jahre werden im Bezug auf Kryptowährungen sehr spannend.

Literaturverzeichnis

Adr19. Adriana Hamacher: *Ethereum's roadmap isn't ambitious enough*, 2019. https://decrypt.co/6136/vulcanize-rick-dudley-ethereum-roadmap-makerdao-polkadot.

Ame16. Ameer Rosic: *Cryptocurrency Wallet Guide: A Step-By-Step Tutorial*, 2016. https://blockgeeks.com/guides/cryptocurrency-wallet-guide/.

Ant16. Antony Lewis: *A gentle introduction to Ethereum*, 2016. https://bitsonblocks.net/2016/10/02/gentle-introduction-ethereum/.

Art19. Arthur Webb: *Top 5 Best Cryptocurrency Wallets 2019, Everything You Need To know*, 2019. https://ripplecoinnews.com/top-5-best-cryptocurrency-wallets.

AW18. Antonopoulos, Andreas M. und Gavin A. Wood: *Mastering Ethereum: Building smart contracts and DApps*. 2018. https://github.com/ethereumbook/ethereumbook.

Bru19. Bruce Schneier: *There's No Good Reason to Trust Blockchain Technology*, 2019. https://www.wired.com/story/theres-no-good-reason-to-trust-blockchain-technology/.

BT. Blockchain-Team: *Charts*. https://www.blockchain.com/de/charts.

Coi19. Coindesk: *How Ethereum Mining Works*, 2019. https://www.coindesk.com/learn/ethereum-101/ethereum-mining-works.

Con. ConsenSys: *Ethereum Smart Contracts Best Practices*. https://consensys.github.io/smart-contract-best-practices/known_attacks/.

Disa. District0x Team: *Ethereum vs. Ether*. https://education.district0x.io/general-topics/understanding-ethereum/ethereum-vs-ether/.

Disb. District0x Team: *What is Ethereum ?* https://education.district0x.io/general-topics/understanding-ethereum/what-is-ethereum/.

ET. Etherbasics-Team: *Die Phasen von Ethereum*. https://etherbasics.com/basics/ethereum-phasen/.

Eth. EthHub Team: *Proof of Stake*. https://docs.ethhub.io/ethereum-roadmap/ethereum-2.0/proof-of-stake/.

Fai18. Faith Obafemi: *Jeffrey Wilcke — Building Decentralized Applications with Global Adoption*, 2018. https://cryptocoremedia.com/jeffrey-wilcke-decentralized/.

Fin. Finanzen.net: *Kryptowährung kaufen – diese Möglichkeiten gibt es*. https://www.finanzen.net/ratgeber/kryptowaehrung/kryptowaehrung-kaufen.

Gar19. Gareth Jenkinson: *Ethereum Classic 51% Attack — The Reality of Proof-of-Work*, 2019. https://cointelegraph.com/news/ethereum-classic-51-attack-the-reality-of-proof-of-work.

Gav14. Gavin Wood: *Ethereum: A Secure Decentralised Generalised Transaction Ledger*, 2014. https://ethereum.github.io/yellowpaper/paper.pdf.

Max16. Max Biederbeck: *Der DAO-Hack: Ein Blockchain-Krimi aus Sachsen*, 2016. https://www.gq-magazin.de/auto-technik/article/wie-aus-dem-hack-des-blockchain-fonds-dao-ein-wirtschaftskrimi-wurde.

Mem19. Memo Özbek: *Was sind ERC20 Token ?*, 2019. https://decentralbox.com/was-sind-erc20-token/.

Oli18. Oliver Dale: *Best Ethereum Wallets 2019: Hardware vs Software vs Paper*, 2018. https://blockonomi.com/best-ethereum-wallets/.

Osm18. Osman Gazi Güçlütürk: *The DAO Hack Explained: Unfortunate Take-off of Smart Contracts*, 2018. https://medium.com/@ogucluturk/the-dao-hack-explained-unfortunate-take-off-of-smart-contracts-2bd8c8db3562.

Pre17. Preethi Kasireddy: *How does Ethereum work, anyway?*, 2017. https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369.

Sol. Solidity Documentation Team: *Introduction to Smart Contracts*. https://solidity.readthedocs.io/en/v0.5.13/introduction-to-smart-contracts.html.

TAN19. TANYA: *What is an ERC20 token?*, 2019. https://support.blockchain.com/hc/en-us/articles/360027491872-What-is-an-ERC20-token-.

Vit13. Vitalik Buterin: *A Next-Generation Smart Contract and Decentralized Application Platform*, 2013. https://github.com/ethereum/wiki/wiki/White-Paper.

Vit15. Vitalik Buterin: *Merkling in Ethereum*, 2015. https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/.

WF19. Wilkens, Robert und Richard Falk: *Smart Contracts: Grundlagen, Anwendungsfelder und rechtliche Aspekte*. essentials. Springer Fachmedien Wiesbaden GmbH and Springer Gabler, Wiesbaden, 1. Auflage 2020 Auflage, 2019.