Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

BACHELORARBEIT

Blockchain-Technologie im Online Advertising

vorgelegt von

Daniel Braun

Fakultät:	Institut für Wirtschaftsinformatik
Studiengang:	Wirtschaftsinformatik
Matrikelnummer:	6922337
Betreuer:	Michael Palk
Erstgutachter:	Prof. Dr. Stefan Voß
	Institut für Wirtschaftsinformatik
Zweitgutachter:	Dr. Kai Brüssau
	Institut für Wirtschaftsinformatik

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

2 Blockchain 1.0 - Wie Bitcoin funktioniert

Mit Fortschritt im Bereich Kryptographie begann auch das Interesse von Forschern an digitalen Währungen. Das Problem dieser frühen Projekte bestand jedoch darin, dass sie einen sogenannten *Central Point of Failure*, also eine zentralisierte Schwachstelle besaßen. Beispielsweise könnten die Konten von Nutzern zwar kryptografisch gesichert, jedoch von zentralen Stellen wie Banken verwaltet werden müssen.
Ein wichtiges Problem, welches es mithilfe von Geld zu lösen gilt, ist das sogenannte *Double Spending Problem*. Es muss durch gewisse Mechanismen verhindert werden, dass bösartige Akteure die selben Geldwerte für mehrere Transaktionen verwenden. Bei physischem Geld, also Geldscheinen, Münzen, etc. verhindern komplexe Drucktechniken die Verbreitung von Falschgeld und dadurch das ein Geldschein nur einmal existieren kann, ist dieser nur für eine Transaktion zu verwenden.
Versucht man nun diese Geldwerte gänzlich digital zu verwalten, so liegt die Verantwortung für eine korrekte Beobachtung und Verwaltung bei einer zentralen Stelle wie einer Bank. Diese könnte als Angriffsstelle für Antagonisten dienen und stellt somit eine Gefahr für das System dar.
Dieses Kapitel beschäftigt sich mit der traditionellen Funktionsweise von Geld und wie mithilfe eines dezentralen Systems ein zentraler Fehlerpunkt vermeidet werden kann.

2.1 Funktionsweise von Geld

Mankiw und Taylor (2018) bezeichnen Geld als ein Bündel von Aktiva, das die Menschen in einer Volkswirtschaft regemäßig dazu verwenden, Waren und Dienstleistungen von anderen Menschen zu erwerben.
Es erlaubt den Parteien einem Tauschgeschäft, bei dem beide Seiten mit dem Gut des Tauschpartners zufrieden sein müssen, zu entgehen und ermöglicht stattdessen eine effiziente Allokation von Ressourcen. Zugleich stellt Geld sicher, dass das eigene Kapital den Wert auch in Zukunft behält.
Damit ein Handelsgut als Geld angesehen werden kann, muss es drei Funktionen erfüllen können
Fundamental ist, dass das Handelsgut generell als Tausch- bzw. Zahlungsmittel akzeptiert wird. Theoretisch könnte man versuchen sein Abendessen mit dem eigenen Fahrrad zu bezahlen, doch kommt man in der Praxis mit dieser Strategie nicht weit.
Des Weiteren muss das Tauschgeschäft als Recheneinheit fungieren können. Dies ist notwendig, da anhand Dessen die relativen Preise anderer Waren in der Marktwirtschaft ermittelt werden müssen.

Zuletzt muss sichergestellt sein, dass das Handelsgut wie bereits erwähnt in Zukunft auch seine Kaufkraft behält. Jemand, der es als Zahlungsmittel akzeptiert muss sich darauf verlassen können, dass es auch für zukünftige Geschäfte verwendet werden kann.
Bei den Geldformen unterscheidet man zwischen Warengeld und Rechengeld. Diese unterscheiden sich in ihrem intrinsischen Wert, also darin, ob sie auch außerhalb von Tauschgeschäften einen Nutzen finden. Ein Beispiel für Warengeld ist Gold, welches neben Tauschgeschäften auch industriell verarbeitet werden kann.
Papiergeld hingegen bietet abseits des Tauschgeschäftes keinen Nutzen für den Besitzer. Um trotzdem den Wert des Geldes gewährleisten zu können, wird es von Seiten des Staats als universelles Zahlungsmittel in der jeweiligen Marktwirtschaft bestimmt.
Eine weitere wichtige Rolle im Finanzsystem nehmen Zentralbanken ein. Sie überwachen das Bankensystem und steuern über eine geeignete Geldpolitik das Geldangebot auf dem Markt.
Durch das Drucken von Geld und den anschließenden Kauf von Wertpapieren können sie das Geldangebot erhöhen. Um es wiederum zu verringern, verkaufen sie Wertpapiere und nehmen das erhaltene Geld aus dem Umlauf.

Eine Währung, die zum Verwalten und Versenden von monetärem Wert dient, hat drei technische Anforderungen zu erfüllen:

1. Sicherstellung des Wertes, also die Authentizität
2. Garantie dafür, dass die selbe Währung nicht mehr als einmal verwendet werden kann (Double Spending)
3. Zugang zur Währung nur für befugten Besitzer

TODO

2.2 Notwendigkeit für Blockchain-Technologie

2.3 Theorie der Blockchain-Technologie am Beispiel von Bitcoin

Auch wenn es andere Projekte für dezentrale Währungen wie B-Money und Hashcash gab, begann der Aufschwung digitaler Währungen im Jahr 2008 mit der Veröffentlichung des Bitcoin-Whitepapers *Bitcoin: A Peer to peer Electronic Cash System*. Diese Publikation wurde, von einer bis heute unbekannten Person, unter dem Namen *Satoshi Nakamoto* veröffentlicht und kombinierte Technologien ihrer Vorgänger. Statt einer zentralen Verwaltungsstelle handelt es sich bei Bitcoin um ein dezentrales Peer-to-peer Netzwerk zwischen den Nutzern des Bitcoin-Protokols. Außerdem werden Vermögenswerte nicht durch

klassischer Münzen auf einem Konto repräsentiert, sondern durch vergangene Transaktionen in einem dezentralen und öffentlichen Transaktionsbuch, dem sogenannten *Ledger* impliziert. Aufgrund dieser Eigenschaften besteht keine zentrale Angriffsfläche für bösartige Akteure und jeder Akteur im Netzwerk hat Kenntnis über alle Transaktionen. Die folgenden Untersektionen beschäftigen sich mit der Verwaltung und dem Zugang für Nutzer, die Funktionsweise von Transaktionen sowie die Art und Weise, wie die verschiedenen Akteure im Netzwerk zu einem gemeinsamen Konsens kommen.

2.3.1 Keys und Adressen

Als Kryptographie bezeichnet man Verfahren zur Verschlüsselung von Informationen, die schon von den Nazis im zweiten Weltkrieg genutzt wurden. Mithilfe von Maschinen, den sogenannten *ENIGMA*, verschlüsselten sie wichtige strategische Informationen wie die Aufenthaltsorte von Truppen oder taktische Befehle, die anschließend per Funk überbracht wurden.
Kryptographische Verfahren folgten zu der Zeit dem Prinzip *Security by Obscurity*, nach dem die Sicherheit eines Verschlüsselungsverfahrens davon abhängig ist, ob die Funktionsweise dieser bekannt ist. Dies hatte zur Folge, dass im Falle der Nazis, deren ENIGMA-Code im Jahr 1941 vom englischen Mathematiker *Alan Turing* und seinem Team gelöst werden konnte.
Im Jahr 1976 stellten *Diffie* und *Hellman* die bis dahin unbekannte asymmetrische Verschlüsselung vor, bei der jede Partei ein Schlüsselpaar, bestehend aus privatem und öffentlichem Schlüssel, besitzt. Derartige Verfahren sind heutzutage der Standard und werden auch im Bitcoin-System verwendet.
Für Bitcoin wird ein Paar aus Schlüsseln erzeugt. Dieses Paar besteht aus dem privaten Schlüssel (private key), welcher nur dem Besitzer bekannt ist und zum Signieren von Transaktionen nötig ist. Aus diesem wird durch die Verwendung von Hashing-Verfahren ein öffentlicher Schlüssel (public key) abgeleitet, mit dem Bitcoins empfangen werden können.
Außerdem kann aufgrund der mathematischen Abhängigkeit zwischen den Schlüsseln eine durch den privaten Schlüssel signierte Transaktion mithilfe des öffentlichen Schlüssels verifiziert werden. Dies geschieht, indem der Absender die Transaktion mit seinem privaten Schlüssel signiert und die Authentizität der Signatur mithilfe des öffentlichen Schlüssels von anderen Akteuren des Netzwerks verifiziert wird. Um Begünstigter einer Transaktion zu sein, muss man eine Adresse besitzen und diese an andere Nutzer des Netzwerks propagieren. Um Jene zu erzeugen, wird der öffentliche Schlüssel genutzt, welchen man nicht wieder aus der Adresse rekonstruieren kann.

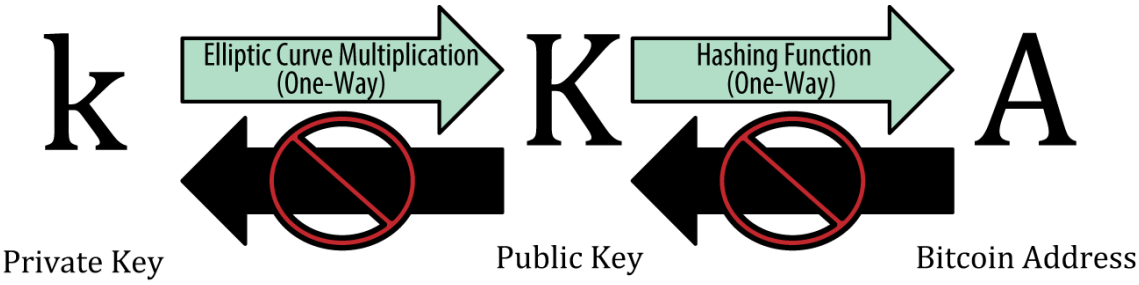

Abbildung 2.1.: Generierung der Schlüssel bzw. Adressen aus dem jeweiligen Vorgänger

Private Keys

Ein privater Schlüssel besteht aus einer Zahl von 256 zufälligen Bits. Er wird zum Signieren von Transaktionen und für den Zugriff auf ein Guthaben benötigt. Ohne privaten Schlüssel verliert man als Besitzer von Bitcoin auch den Zugriff auf das eigene Guthaben. Um einen privaten Schlüssel generieren zu können, benötigt man eine sichere Quelle für SZufälligkeit". In anderen Worten: Die Wahl der zufälligen Zahl darf nicht vorhersehbar sein. Dazu verwendet die Bitcoin-Software den Random Number Generator des verwendeten Betriebssystems kombiniert mit einem menschlichen Input, wie dem Bewegen der Maus. Mithilfe des Generators erzeugt man einen zufälligen String, welcher *mehr* als 256 Bits hat. Diesen lässt man anschließend durch den SHA256 Hash-Algorithmus laufen und prüft, ob die resultierende Zahl kleiner ist, als die vom Bitcoin-Protokol gewählte Konstante n ($n = 1.1578 * 10^{77}$).

Public Keys

Um einen öffentlichen Schlüssel aus dem Privaten generieren zu können, benötigt man ein kryptografisches Verfahren, welches eine Rekonstruktion des Privaten aus dem öffentlichen Schlüssel nicht zulässt. Das vom Bitcoin-Protokol verwendete Verfahren wird *Elliptic Curve Cryptography* gennant und bedient sich an den Eigenschaften einer Ellipse. Um einen öffentlichen Schlüssel zu generieren, wählt man einen Punkt, den sogenannten Generatorpunkt, auf der Ellipse und Multipliziert diesen mit dem vorher generierten privaten Schlüssel. Eine Multiplikation kann auch als Addition einer Zahl mit derselben betrachtet werden. Um den Punkt G auf der Ellipse mit sich selbst zu addieren, zieht man an diesem die Tangente und berechnet den Schnittpunkt von Ellipse und der gezogenen Tangente. Anschließend spiegelt man den Punkt an der x-Achse, erhält 2G. Diese Addition führt man so oft aus, wie der 256 Bit lange private Schlüssel groß ist, sodass man am Ende einen Punkt (x,y) erhält, welcher als öffentlicher Schlüssel genutzt werden kann. Diesen generierten Schlüssel kann man veröffentlichen, denn aus ihm lässt sich nicht schließen,

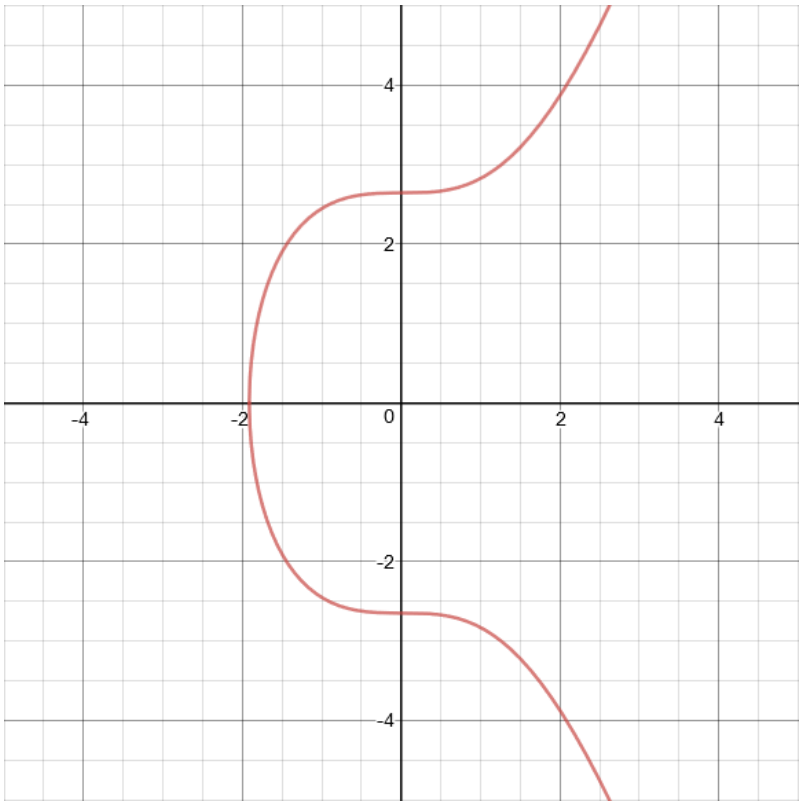

Abbildung 2.2.: Die von Bitcoin verwendete Ellipse mit der Funktion $y^2 = x^3 + 7$
TODO: Bessere Grafik

mit welchem Faktor der Generatorpunkt multipliziert wurde.

Bitcoin Adressen

Eine Adresse ist ein aus dem öffentlichen Schlüssel generierter String aus Buchstaben und Zahlen, der den Besitzer des Schlüssels zu einem potentiellen Empfänger einer Transaktion macht. Beim diesem muss es sich allerdings nicht zwangsläufig um eine Person handeln, denn auch Organisationen, geschriebene Skripte, etc. kommen als *abstrakter* Empfänger in Frage.
So wie der Öffentliche aus dem privaten Schlüssel erzeugt wird, wird die Bitcoin Adresse aus dem öffentlichen Schlüssel mithilfe von Hashing-Algorithmen erzeugt. Die verwendeten Algorithmen, welche nacheinander auf den öffentlichen Schlüssel angewendet werden, heißen SHA-256 und RIPEMD160.
Da Verschlüsselungsalgorithmen einen Grundbaustein für Blockchain-Technologie darstellen, wird ihre Funktionsweise im Folgenden beispielhaft anhand des SHA-256-Algorithmus erläutert.

2.3.2 Einschub: SHA-256

Ein Hashalgorithmus ist eine mathematische Funktion, die einen Input entgegennimmt und einen Output fester Größe, den sogenannten Hashwert erzeugt. Ein Algorithmus sollte idealerweise folgende Eigenschaften bieten:

- Für einen gegebenen Input immer denselben Output generieren
- Nur in eine Richtung berechenbar sein
- Durch geringe Änderungen am Input einen völlig anderen Output generieren

Hashfunktionen haben vielerlei Anwendungsmöglichkeiten im Bereich der Datensicherheit und eine verwendete Hashfunktion im Bitcoin-Protokoll ist die bereits erwähnte SHA256-Funktion, welche u.A. von der NSA entwickelt wurde. Das folgende Unterkapitel orientiert sich an Dang (2015), ist allerdings für das Verständnis nachfolgender Kapitel nicht zwingend erforderlich.

Definitionen

Bevor der Algorithmus erläutert werden kann, ist es notwendig folgende Operationen, sowie benötigte Konstanten zu definieren. Die Operationen werden auf Zahlen in Binärform bitweise angewendet.

Tabelle 2.1.: Benötigte Operationen

+	Addition in Mod 2^{32}
XOR	Vergleichender Operator $\oplus$. Ergebnis ist True(1), wenn genau einer der beiden Inputs True ist. Andernfalls wird False(0) ausgegeben. Beispiel: $0 \oplus 1 = 1$
Rotation Right	Verschiebung der Bits um n Stellen nach rechts. Bei Overflow werden Bits wieder vorne angefügt. Beispiel: $ROTR_1(011) = 101$

Shift Right	Ähnlich wie die Rotation, nur dass Bits an letzter Stelle wegfallen und eine Nullen vorne angefügt werden. Beispiel: $$SHR_1(011) = 001$$
Choice	Nimmt drei Zahlen gleicher Länge entgegen und entscheidet anhand der Bits des ersten Parameters, welches Bit der jeweils anderen Parameter übernommen werden soll. $$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$$ Beispiel: $Ch(110,001,101) = 001$
Majority	Nimmt drei Zahlen gleicher Länge entgegen und übernimmt an jedem Bit denjenigen Wert, der zwischen den Inputs am häufigsten auftaucht. $$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$ Beispiel $Maj(110,001,101) = 101$
σ_0	σ_0 und die folgenden Funktionen sind definierte Folgen der oben definierten Operationen. $$\sigma_0(x) = ROTR_7(x) \oplus ROTR_{18}(x) \oplus SHR_3(x)$$
σ_1	$$\sigma_1(x) = ROTR_{17}(x) \oplus ROTR_{19}(x) \oplus SHR_10(x)$$
Σ_0	$$\Sigma_0(x) = ROTR_2(x) \oplus ROTR_{13}(x) \oplus ROTR_{22}(x)$$
Σ_1	$$\Sigma_1(x) = ROTR_6(x) \oplus ROTR_{11}(x) \oplus ROTR_{25}(x)$$

Außerdem müssen noch zwei Listen mit Konstanten definiert werden. Die Liste K beinhaltet die ersten 32 Bit der Nachkommastellen der Kubikwurzeln der ersten 64 Primzahlen. H die ersten 32 Bit der Nachkommastellen der Quadratwurzeln der ersten 8 Primzahlen. Die genauen Werte haben keine sonderliche Bedeutung, sondern lediglich die scheinbare Zufälligkeit ist von Relevanz. Indem man diese berechenbaren Zahlen nimmt, minimiert sich die Wahrscheinlichkeit für eine mögliche Backdoor im Algorithmus.

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4
ab1c5ed5 d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe
9bdc06a7 c19bf174 e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f
4a7484aa 5cb0a9dc 76f988da 983e5152 a831c66d b00327c8 bf597fc7
c6e00bf3 d5a79147 06ca6351 14292967 27b70a85 2e1b2138 4d2c6dfc
53380d13 650a7354 766a0abb 81c2c92e 92722c85 a2bfe8a1 a81a664b
c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070 19a4c116
1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7
c67178f2
```

Listing 2.1: Liste K von Konstanten

```
H0 = 6a09e667
H1 = bb67ae85
H2 = 3c6ef372
H3 = a54ff53a
H4 = 510e527f
H5 = 9b05688c
H6 = 1f83d9ab
H7 = 5be0cd19
```

Listing 2.2: Liste H mit den Arbeitsvariablen H0 - H7

Die Zahlen befinden sich zu diesem Zeitpunkt in Hexadezimal-Form, müssen aber in Binärzahlen umgewandelt werden.

Vorbereitung der Nachricht

Bevor eine Nachricht verarbeitet werden kann, muss sie in eine für den Algorithmus brauchbare Form gebracht werden. Da der SHA-256 mit Zahlen in Binärform arbeitet, müssen Inputs in Stringformat über die ASCII-Tabelle in Binärzahlen umgewandelt werden. So wird z.B. aus dem String 'ab' die Zahl 01100001 01100010. Anschließend fügt man eine Eins und k-viele Nullen hinzu, dass ein Vielfaches von 512 abgezogen 64 her-

auskommt. Dang (2015) definiert diese Voraussetzung durch

$$l + 1 + k \equiv 448 Mod 512$$

mit l als Länge der Nachricht. Anschließend werden 64 Bits hinzugefügt, welche die Zahl l repräsentieren, im Fall 'ab' wäre das die 16 in 64-Bit Repräsentation. Die resultierende N*512-Bit Zahl stellt den Input für die eigentliche Verschlüsselung dar.

Aufteilung der Nachricht in Blöcke

Die resultierende Zahl wird anschließend in Nachrichtenblöcke M_1 - M_N der Länge 512-Bit und diese wiederum in 16 32-Bit-Blöcke M_{i0} - M_{i15} aufgeteilt, die man auch Wörter nennt. Der String 'ab' resultiert in einer 512-Bit-Zahl, sodass an dieser Stelle nur eine Aufteilung in 16 32-Bit-Blöcke nötig wäre.

Algorithmus

Im folgenden soll der Algorithmus anhand eines Pseudo-Codes erläutert werden. Zu beachten ist, dass die Liste k, sowie die Variablen H0-H7 global definiert seien.

```
For i=1 to N:
  w = []
  For t=0 to 63:
    if t <= 15:
      w.push(M[i][t])
    else:
      a = σ1(w[t-2]) + w[t-7] + σ0(w[t-15]) + w[t-16]
      w.push(a)
  a = H0
  b = H1
  c = H2
  ...
  h = H7
  For j=0 to 63
    T1 = h + Σ1(e) + Ch(e,f,g) + k[j] + w[j]
    T2 = Σ0(a) + Maj(a,b,c)
    h = g
    g = f
    e = d + T1
    d = c
```

```
21        c = b
22        b = a
23        a = T1 + T2
24      H0 = a + H0
25      H1 = b + H1
26      ...
27      H7 = h + H7
28    result = ''.concat(H1).concat(H2). ... .concat(H7)
```

Listing 2.3: Pseudocode zu SHA256

Es wird zu Beginn ein leerer String initialisiert, der am Ende das Ergebnis des Algorithmus darstellt [1].
Anschließend wird die erste Schleife gestartet, die für jeden der N 512-Bit-Blöcke durchlaufen wird. Zu Beginn jedes Schleifendurchlaufs wird eine leere Liste *w* initialisiert, die während des Durchlaufs gefüllt wird [2-3].
Für das Füllen der Liste wird eine weitere Schleife initialisiert, die von 0 bis 63 läuft und in den ersten 16 Durchgängen lediglich die 16 Wörter aus M[i] in die bisher leere Liste w einfügt [4-6].
Ab Schleifendurchgang 16 werden die übrigen Einträge mithilfe der vorher definierten Funktionen berechnet, sodass man am Ende eine Liste mit 64 Einträgen hat [7-9].
Anschließend werden die sogenannten Arbeitsvariablen a bis h mithilfe der vorher definierten H0-H7 initialisiert [10-14].
Die folgende Schleife ist die, in der die eigentliche Kompression stattfindet. Sie läuft erneut von 0 bis 63 [15].
Mithilfe der vorher Definierten Funktionen und der Arbeitsvariablen werden die temporären Variablen T1 und T2 ermittelt. Für T1 spielen zusätzlich die Listen k, welche die Kubikzahlen der ersten 64 Primzahlen enthält, und die Liste w, welche die in Zeile 4 bis 9 generierten Wörter enthält, eine Rolle [16-17].
Anschließend werden die Arbeitsvariablen mithilfe der Temporären neu gesetzt, sodass man nach 64 Durchläufen scheinbar willkürliche Zahlen erhält [18-24].
Man addiert schließlich die Arbeitsvariablen und ursprünglichen Variablen H0-H7 und kettet diese aneinander. Bei einer kleinen Nachricht wie dem 'ab', welche lediglich einen 512-Bit-Block benötigt, wäre dies bereits das Ergebnis des Algorithmus. Sollte der Input ein Vielfaches von 512-Bit benötigen, wird die äußerste Schleife mehrfach durchlaufen und die Variablen H0-H7 in den Zeilen 25-28 nicht mehr mithilfe der initial definierten H0-H7, sondern mit denen der vorangegangenen Iteration berechnet.
Unabhängig von der Länge des Inputs erhält man so immer einen Output von 256 Bits, da die Arbeitsvariablen jeweils eine Länge von 32 Bits haben.

2.3.3 Wallet

Eine Wallet ist ein Programm, welches als Interface zwischen Bitcoin-Netzwerk und dem Nutzer dient. Dessen Funktionen beinhalten die Verwaltung der Schlüssel, das Berechnen des Guthabens und das Signieren von Transaktionen. Eine Wallet ist, im Gegensatz zu einer physikalischen Geldbörse nicht für das Halten von Münzen, sondern zur Verwaltung der privaten Schlüssel zuständig. Wie das Berechnen des "Guthabens"geschieht, wird im Unterkapitel *Transaktionen erläutert*.
Man unterscheidet zwischen nicht-deterministischen und deterministischen Wallets. Die erste Variante kann man sich als Korb vorstellen, in dem vorher zufällig generierte private Schlüssel in großer Anzahl gelagert sind. Dabei erzeugt ein Privater einen öffentlichen Schlüssel, der wiederum eine Adresse erzeugt (siehe Kapitel *Keys und Adressen*). Um die eigene Pseudonymität zu schützen ist es empfehlenswert, einen Key nur ein Mal zu benutzen. Aufgrund der hohen Anzahl angesammelter Keys und der damit verbundenen Datensicherung ist diese Art Wallet heute nicht mehr der Standard.
Die fortgeschrittenste Form einer Deterministischen ist die sogenannte BIP32-Wallet,

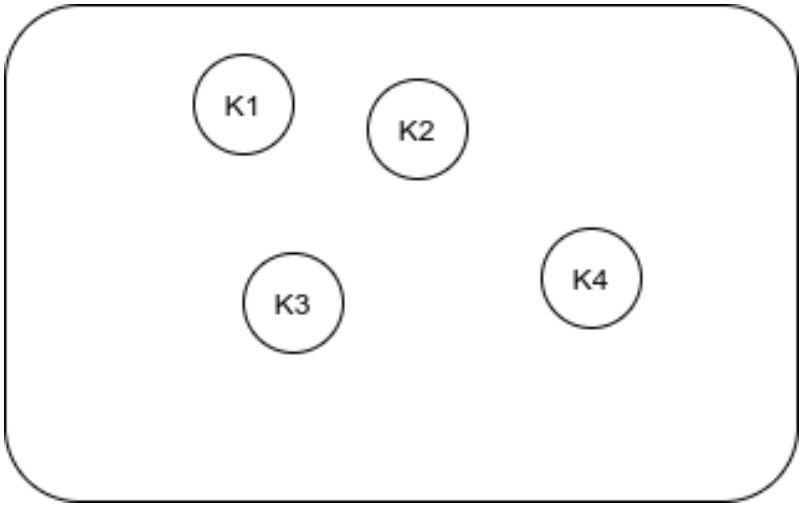

Abbildung 2.3.: Nicht-deterministische Wallet

welche 2 nützliche Eigenschaften aufweisen kann (BIP steht für *Bitcoin Improvement Proposal* und bezeichnet eine nachträgliche Ergänzung zum Bitcoin-Ökosystem).
Diese werden in Buterin (2013) als *Master Public Key Property* und *Hierarchy Property* bezeichnet. Die Master Public Key Property beschreibt die Möglichkeit, aus einem Master Private einen Master Public Key zu generieren, der wiederum alle öffentlichen Schlüssel und deren Adressen erzeugen kann. Dazu berechnet man den sogenannten Offsets, indem man den gewünschten Index und den Master Public Key addiert und das Ergebnis als Input für eine Hashfunktion verwendet. Anschließend addiert man Offset und Master Public

Key und erhält den öffentlichen Schlüssel am Index.

$$offset = SHA256(index + masterPubKey)$$

$$pubKey_{index} = offset + masterPubKey$$

Dies geht analog genauso mit dem Master Private Key. Aufgrund dieser Eigenschaft ist es möglich, den Master Public Key ungeschützt zu lagern und sogar an dritte Parteien herauszugeben, ohne dass diese Zugriff auf das Guthaben erhalten.
Die Hierarchieeigenschaft wird im Kontext einer Organisation mit verschiedenen Organisationszweigen interessant. Ein Geschäftsführer könnte so den unterschiedlichen Geschäftszweigen seines Unternehmens Schlüsselpaare zuweisen, wodurch diese die Verfügungsgewalt über das Eigene und Guthaben von Unterstellen erhalten. Gleichzeitig behält der Geschäftsführer die absolute Kontrolle über alle Schlüssel, da er im Besitz der Master Keys ist. Anders als bei einer nicht-deterministischen Wallet müssen nicht mehr die Priva-

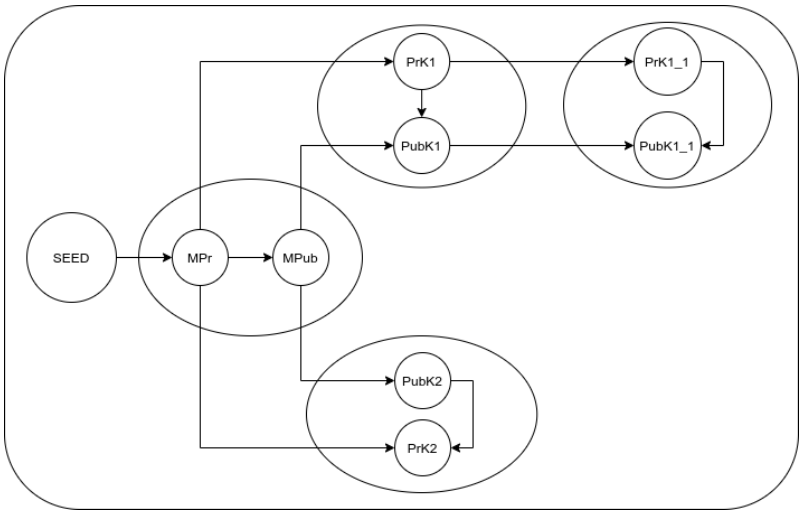

Abbildung 2.4.: BIP32-Wallet

te Keys selbst, sondern lediglich der *Seed* gesichert werden. Dieser ist seit dem BIP39/44 eine kurze Liste von für den Menschen leserlichen Worten. Diese können mithilfe eines speziellen Dictionaries in Hex-Zahlen umgewandelt werden, aus denen schließlich der Master Private Key erzeugt wird.

2.3.4 Transaktionen

Damit Bitcoin die Funktion des Zahlungs- bzw. Tauschmittels erfüllen kann, müssen Transaktionen vom Netzwerk ermöglicht werden. In einem *Distributed Ledger*, einer Art dezentraler Datenbank, werden alle Transaktionen im Netzwerk aufgezeichnet. Da es sich um ein gänzlich digitales und dezentrales Netzwerk handelt, sind spezielle Datenstruktu-

ren und Mechanismen nötig.

Transaction Outputs

Das Guthaben einer Wallet befindet sich nicht an einem Ort, sondern wird aus *Unspent Transaction Outputs*, kurz UTXO zusammengesetzt, welche alle verfügbaren Gutschriften aus vergangenen Transaktionen darstellen. Eine Wallet scannt die Blockchain nach allen Outputs, welche mit den Keys des Besitzers assoziiert sind und errechnet damit das gesamte verfügbare Guthaben. Außerdem speichert sie die nötigen Referenzen für den Zugriff auf die UTXO für den späteren Gebrauch in einer Datenbank ab.
UTXO¸ sind diskrete, sowie unteilbare Einheiten und können demnach nur in ihrer Gesamtheit genutzt werden. Man betrachte die folgende Abbildungen: Für die gewünschte

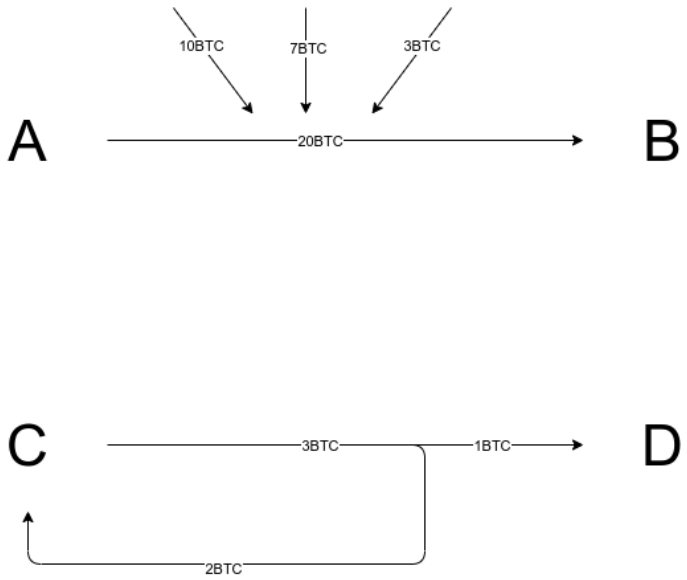

Abbildung 2.5.: Transaktionen mit unteilbaren UTXO

Transaktion von A nach B werden 20 BTC benötigt, die von der Wallet aus verfügbaren UTXO zusammengefügt werden. Für das optimale Zusammensammeln von Outputs ist die Wallet zuständig.
C hat für die gewünschte Transaktion von 1 BTC nur 3 BTC zur Verfügung, weshalb diese in ihrer Gesamtheit aufgebraucht werden müssen. Die übrigen 2 BTC gehen jedoch nicht verloren, sondern landen lediglich wieder als Wechselgeld bei C. Die gewünschte Transaktion hat also 2 Outputs.
Jeder Output besteht aus 2 Komponenten: Der Anzahl von BTC und einem sogenannten *Locking Script*, welches nur mithilfe des assoziierten private Keys gelöst werden kann.

Transaction Inputs

Das Gegenstück zu den Outputs sind Inputs, über welche die benötigten UTXO für eine Transaktion gesammelt werden. Sie bestehen aus:

- Einer ID, über die eine bestimmte Transaktion referenziert wird
- Vout, einem Index über den auf bestimmte UTXO der zuvor referenzierten Transaktion zugegriffen wird
- Dem Unlocking Script, welches das Locking Script des ausgewählten UTXO löst und mithilfe des privaten Schlüssels erzeugt wird
- Einer Sequence, einem optionalen Feld zur Sperrung der Outputs für eine bestimmte Zeit

Sowohl Input, als auch Output werden serialisiert und als Byte-Streams im Netzwerk propagiert.

Transaktionskosten

Abhängig von der Auslastung des Netzwerkes wird eine dadurch bestimmte Menge an Transaktionsgebühren nötig, die sowohl als Anreiz für Bitcoin-Miner, als auch Abschreckungsmechanismus gegen Angriffe durch hochfrequentes Senden von kleinen Transaktionen dienen.
Diese sind zwar kein Muss, jedoch bevorzugen die Miner Transaktionen mit höheren Transaktionskosten. Das berechnen von optimalen Transaktionsgebühren wird normalerweise von der Wallet ausgeführt, jedoch können auch manuelle Transaktionen erstellt werden, die ohne Transaktionskosten langsamer oder gar nicht vom Netzwerk verarbeitet werden. Bei einer manuellen Transaktion müssen sowohl Inputs, als auch Outputs definiert werden und die Transaktionsgebühren ergeben sich implizit aus der Differenz jener Komponenten. So wie man keine Transaktionsgebühren angegeben kann, ist auch das versehentliche Angeben von zu hohen Gebühren möglich. Man betrachte erneut die Abbildung 2.5, in der C 3 BTC an D sendet. Die Transaktion hat 2 Outputs, 1 BTC an D und 2 BTC zurück an C. Sollte der Sender vergessen den 2. Output, welcher das Wechselgeld darstellt, zu definieren, dann werden die gesamten 2 BTC als Transaktionsgebühren für die Transaktion genutzt.

Zusammenfassend lassen sich Transaktionen als Sammlungen von Inputs und Outputs definieren, deren Differenz die genutzten Transaktionskosten darstellen.

2.3.5 Die verschiedenen Akteure im Netzwerk

In Nakamoto (2008) nennt Nakamoto das Propagieren von Transaktionen an alle Knoten als ersten Schritt, der zur Instandhaltung des Netzwerkes erforderlich ist. Dieses Kapitel steigt einen Schritt früher ein und befasst sich mit den verschiedenen Akteuren im dezentralen Netzwerk, ihren Rollen sowie den nötigen Schritten zum Eintritt neuer Teilnehmer.

Rollen

Trotz fehlender Hierarchien im Netzwerk nehmen die Teilnehmer abhängig von ihren Funktionalitäten verschiedene Rollen ein. Als *Full Node* bezeichnet man Teilnehmer, die alle der folgenden Funktionalitäten erfüllen:

- Routing Modul - Kommunikation mit dem Netzwerk
- Wallet - Verwaltung von Guthaben
- Miner - Finden neuer Blöcke
- Full Blockchain - Besitz gesamter Blockchain

Diese können selbstständig Transaktionen prüfen, signieren und am Konsens-Algorithmus teilnehmen.
Bis auf das Routing-Modul, ohne das die Kommunikation mit dem Netzwerk nicht möglich ist, können sich die Teilnehmer ein Subset der Funktionalitäten heraussuchen und diese erfüllen. Beispielsweise benötigt eine Lightweight-Wallet kein Mining und auch keine volle Blockchain. Stattdessen lässt sie Transaktionen von einer dritten Partei prüfen und könnte so aufgrund des geringen Speicherbedarfs in einem Internetbrowser laufen.

Bootstrapping neuen Teilnehmers

Nach dem Hochfahren muss ein neuer Teilnehmer zunächst mindestens einen Anderen im Netzwerk finden. Dabei stehen ihm sogenannte *DNS Seeds* zur Verfügung, die nichts anderes sind als Server, die auf Abfrage Adressen von stabil laufenden Teilnehmern zurückgeben und deren Adressen im Bitcoin-Klienten enthalten sind. Alternativ kann ein neuer Knoten auch eine manuelle Verbindung zu einem anderen Teilnehmer eingehen, wenn ihm dessen Adresse bekannt ist.
Sobald ein anderer Teilnehmer gefunden wurde wird über TCP ein *Handshake* ausgeführt: Der Neue sendet seine Informationen, wie die Version seines Bitcoin-Klienten an den gefundenen Teilnehmer und dieser geht bei Validität der Informationen eine Verbindung mit ihm ein. Anschließend können sie ihnen bekannte Adressen austauschen:

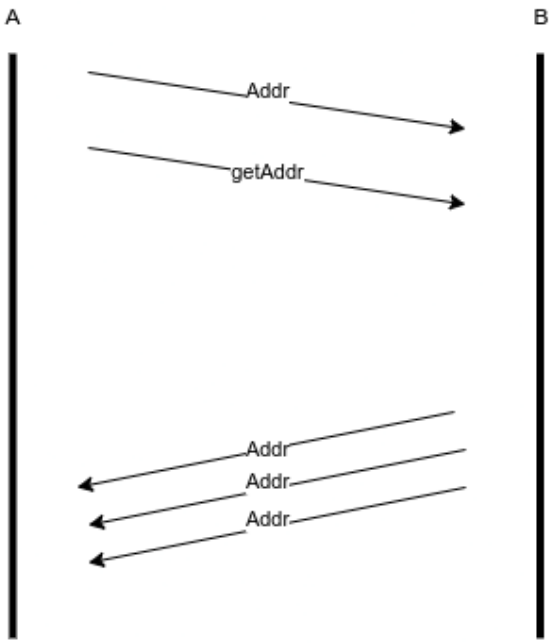

Abbildung 2.6.: Teilnehmer tauschen nach erfolgreicher Verbindung Adressen aus

Ebenso wie sie Adressen miteinander austauschen, werden auch Informationen über den Zustand der Blockchain ausgetauscht. Sollten dem neuen Teilnehmer Blöcke fehlen, versucht er diese von seinen neuen Nachbarn anzufordern, um seine eigene Blockchain zu synchronisieren. Dabei wird der Aufwand auf die verschiedenen Nachbarn aufgeteilt, um einzelne Teilnehmer nicht zu überlasten. Es ist dasselbe Prinzip, wie beim schon länger praktizierten File-Sharing.

2.3.6 Blockchain

Die Blockchain ist die Kerntechnologie des gesamten Protokolls und stellt im Kern eine Datenstruktur mit einer verketteten Liste von Blöcken dar, die wiederum Transaktionen enthalten. Die Verkettung erfolgt dadurch, dass jeder Block einen vorher Erzeugten referenziert. Genauer gesagt enthält der neue Block den SHA256-Hash des Vorgängerblocks, mit dem dieser eindeutig identifizierbar ist. Dadurch wird der Hashwert des Nachfolger verändert und dies wird so für alle folgenden Generationen fortgesetzt. Der daraus ent-

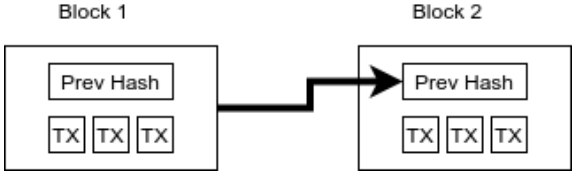

Abbildung 2.7.: Neuer Block referenziert alten Block

stehende Vorteil ist, dass ein Angreifer bei Veränderung eines vergangenen Blocks die

Hashwerte aller Nachfolger neu berechnen muss. Wieso dies mit sehr hohem Rechenaufwand verbunden ist, wird später näher erläutert.

Datenstruktur

Die Datenstruktur des Blocks besteht aus mehr Komponenten als dem Vorgänger-Hash und den Transaktionen:

```
Block {
  Block Size;
  Block Header;
  Transaction Counter;
  Transactions;
}

Block Header {
  Version;
  Prev Block Hash;
  Merkle Root;
  Timestamp;
  Difficulty Target;
  Nonce;
}
```

Listing 2.4: Datenstruktur des Blocks

Der zuvor angesprochene Hashwert wird nicht aus dem gesamten Block, sondern dem *Block Header* errechnet, der Meta-Daten zum Block enthält. Die Komponenten *Merkle Root*, *Difficulty Target* und *Nonce* werden an geeigneter Stelle erläutert.
Abgesehen vom Block-Hash lässt sich ein Block auch über die Block-Höhe, was lediglich dem Index des Blocks in der Liste entspricht, identifizieren. Dafür folgt man vom Block aus über die Vorgänger der Blockchain, bis man beim sogenannten *Genesis Block*, dem ersten Block, ankommt. Anschließend zählt man den Abstand und erhält die Block-Höhe. Eine eindeutige Identifikation des Blocks ist hierüber allerdings nicht möglich, denn es kann im Falle von *Forks*, die im Kapitel *Proof-of-Work* erörtert werden, dazu kommen, dass ein Block von mehreren Blöcken als Parent referenziert wird.

Merkle-Tree

Ein Merkle-Tree ist eine binäre Baumstruktur, bestehend aus den Hashwerten der Transaktionen des Blocks als Blätter. Auf jeder Ebene werden die Knoten paarweise gehasht,

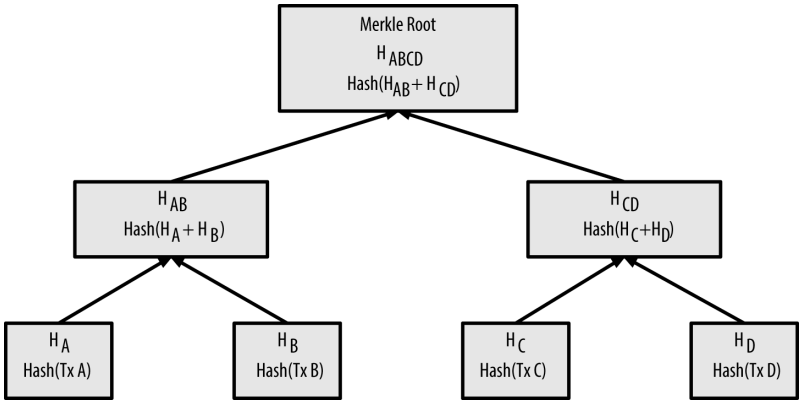

Abbildung 2.8.: Teilnehmer tauschen nach erfolgreicher Verbindung Adressen aus

sodass man als Ergebnis die Wurzel als Identität aller Transaktionen im Block erhält. Aufgrund einer grundlegenden Eigenschaft von Hash-Funktionen, würde eine minimale Änderung an den Transaktionen die Wurzel, und somit den gesamten Block-Hash völlig verändern.

2.3.7 Proof-of-Work

2.3.8 Angriff auf das Netzwerk

3 Blockchain 2.0

3.1 Probleme von Bitcoin und Erweiterung der Konzepte durch Ethereum

3.2 Theoretische Grundlagen von Ethereum

4 Blockchain im Online Advertising

4.1 Wie Online Advertising funktioniert

4.2 Mögliche Verbesserungen mittels Blockchain-Technologie

4.3 Programmierung eines geeigneten Smart Contracts in Solidity

4.4 Beantwortung der Forschungsfrage

4.5 Blockchain 3.0 - Bestehende Probleme und potenzielle Lösungen

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung der Kapitel

5.2 Diskussion der Ergebnisse

5.3 Ausblick

A Anhang

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Cras semper. Integer sapien nulla, consectetuer a, laoreet et, varius quis, mauris. Nunc pharetra tincidunt massa. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Praesent pellentesque mauris at elit. Aliquam consequat suscipit enim. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nunc sapien. Proin hendrerit diam at quam. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Integer vulputate semper nunc. Sed dui. Praesent at sem. Integer elit ipsum, placerat vitae, dictum quis, feugiat sit amet, metus.

A.1 Anhang A

Donec arcu turpis, pretium quis, interdum non, condimentum a, est. Fusce lobortis urna non tellus. Nam leo dui, malesuada non, tempus placerat, congue eget, pede. Mauris porttitor risus quis tortor molestie vehicula. Curabitur tincidunt. In malesuada congue nisi. Nullam et nulla. Curabitur porttitor. Ut molestie sagittis felis. Sed urna libero, ultricies quis, laoreet eget, congue id, metus. Proin ac lorem cursus mauris auctor laoreet. Donec justo. Etiam nunc sem, dapibus sit amet, euismod a, molestie sit amet, mi.

Morbi sollicitudin consequat magna. Vivamus dictum. Nulla non quam. Nam sem tellus, aliquam sed, hendrerit nec, imperdiet ut, augue. Aliquam erat volutpat. Vivamus non ligula sit amet lorem accumsan viverra. Cras mattis libero et ante. Cras massa. Donec fringilla, metus vitae semper condimentum, dolor dui fringilla arcu, et mattis nulla dui vel lectus. Nunc mauris magna, tristique eu, rutrum at, facilisis eu, odio. Nullam congue magna non nisi. Suspendisse viverra, massa non pellentesque scelerisque, risus elit

Hier kommt ein Listing **??**.

```
1   428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
2   d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
3   e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
4   983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
5   27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
6   a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
7   19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
8   748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

```
1   H0 = 6a09e667
```

```
  H1 = bb67ae85
  H2 = 3c6ef372
  H3 = a54ff53a
  H4 = 510e527f
  H5 = 9b05688c
  H6 = 1f83d9ab
  H7 = 5be0cd19
```

bibendum dolor, vitae ultrices lorem neque et erat. Nullam tortor ante, venenatis et, aliquet ac, ornare id, massa. Vivamus urna augue, posuere vitae, sagittis id, porttitor at, arcu. Praesent pharetra rutrum neque. Maecenas tempor ultrices felis. Nulla facilisi. In sed elit aliquet neque malesuada blandit. Nam tempus imperdiet eros. Mauris tincidunt diam eu erat. Phasellus iaculis blandit leo. Nunc augue. Donec dignissim accumsan pede. Ut consequat, eros id accumsan placerat, mi justo ullamcorper pede, id lacinia augue nisi non nibh. Vestibulum eget arcu. Cras pretium, dui eu gravida varius, lectus neque accumsan ligula, eu sodales magna lectus ut nisi. Aliquam vel ante. Ut suscipit porta augue. Suspendisse pellentesque faucibus nisl. Nulla magna tortor, cursus quis, varius quis, hendrerit ut, neque.

Literaturverzeichnis

Buterin, V. (2013). Deterministic wallets, their advantages and their understated flaws. https://bitcoinmagazine.com/articles/deterministic-wallets-advantages-flaw-1385450276/. Letzter Zugriff am 21.02.2021.

Dang, Q. H. (2015). Secure Hash Standard (SHS. http://dx.doi.org/10.6028/NIST.FIPS.180-4/. Letzter Zugriff am 14.02.2021.

Mankiw, N. G., M. P. Taylor (2018). *Grundzüge der Volkswirtschaftslehre* (7. Aufl.). Schäffer-Poeschel Verlag Stuttgart, Stuttgart.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Studiengang Wirtschaftsinformatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg, den _____ Unterschrift: _____