Hochschule für angewandte Wissenschaften
München

Fakultät für Informatik und Mathematik

Bitcoin unter der Lupe

Die technischen Aspekte der revolutionären Kryptowährung genauer betrachtet

Bachelorarbeit

zur Erlangung des akademischen Grades

Bachelor of Science

im Studiengang Informatik

Autor	Marinus Veit
Matrikelnummer:	16432414
Abgabedatum:	31. Dezember 2024
Aufgabensteller:	Prof. Dr. Matthias Güdemann

Eidesstattliche Erklärung
(gemäß § 16, Abs. 10 APO)

Hiermit erkläre ich gemäß der Rahmenprüfungsordnung der Hochschule München, dass ich die vorliegende Arbeit mit dem Titel „Bitcoin unter der Lupe“ selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

..... , den ....M.Veit.. Marinus Veit 2024.12.31 17:36:39 +01'00' .....

Ort Datum Unterschrift Herr Marinus Veit

Kurzfassung

Die vorliegende Arbeit analysiert die technischen Grundlagen von Bitcoin, wie sie in dem Whitepaper „Bitcoin: A Peer-to-Peer Electronic Cash System“ von Satoshi Nakamoto beschrieben sind und bietet einen umfassenden Einblick in diese revolutionäre Kryptowährung.
Zu Beginn werden die kryptografischen Grundlagen von Bitcoin erläutert, darunter Hashfunktionen, Hash-Bäume sowie Schlüssel und Signaturen. Anschließend wird der Aufbau und Ablauf von Bitcoin-Transaktionen beschrieben, einschließlich der Funktionsweise der zugrunde liegenden Skripte. Dabei werden die beiden Standard-Skripte „Pay to Public Key“ (p2pk) und „Pay to Public Key Hash“ (p2pkh) genauer beschrieben.
Ein zentrales Kapitel widmet sich der Blockchain, deren Struktur und Funktionsweise im Detail untersucht werden. Es wird dargelegt, wie Bitcoin durch diese innovative Datenstruktur einen verteilten Konsens in einem genehmigungslosen Peer-to-Peer-Netzwerk erreicht. Abschließend wird das Bitcoin-Netzwerk vorgestellt, wobei die verschiedenen Teilnehmer und ihre jeweiligen Aufgaben im Netzwerk beleuchtet werden.

Inhaltsverzeichnis

Begriffsdefinitionen

Adresse Eine Bitcoin-Adresse ist eine benutzerfreundliche Darstellung des Empfängers, an die Bitcoins gesendet werden können.

Bitcoin Bezieht sich sowohl auf das dezentrale digitale Währungssystem als auch auf die Währungseinheit der Kryptowährung.

Bitcoin-Netzwerk Ein dezentrales Netzwerk von Computern (Knoten), in dem Daten ausgetauscht und validiert werden.

Bitcoin-Protokoll Das Bitcoin-Protokoll definiert die Regeln und Mechanismen, die den Betrieb des Bitcoin-Netzwerks ermöglichen. Es ist in der Bitcoin-Software implementiert.

Block Ein Block ist ein Datensatz in der Blockchain und enthält eine begrenzte Anzahl an Transaktionen. Jeder Block verweist auf einen vorherigen Block, um die Reihenfolge und Integrität der Blockchain zu gewährleisten.

Blockchain Die Blockchain ist eine Kette von Blöcken, die die komplette Transktionshistorie zeitlich geordnet enthält .

BTC Abkürzung für die Währungseinheit von Bitcoin.

Coinbase-Transaktion In einer Coinbase-Transaktion wird der Blockzuschuss, den ein Miner für das erfolgreiche Schürfen eines Blocks erhält, ausgezahlt. Sie ist eine spezielle Transaktion, die keine Eingabe hat und über die neue Bitcoins im Netzwerk geschaffen werden. Die Coinbase-Transaktion dient auch dazu, Transaktionsgebühren aus dem Block an den Miner auszuzahlen.

Double Spending Das Double-Spending-Problem bezeichnet den Versuch, dieselben Bitcoins mehrmals auszugeben.

HASH160 Nachricht wird zuerst mit SHA-256 und anschließend mit RIPEMD-160 gehasht.

HASH256 Nachricht wird zweimal mit der SHA-256 Hashfunktion gehasht. Wird auch Double SHA-256, double-SHA-256 oder SHA-256d genannt.

Input Referenziert die Bitcoins, die in einer Transaktion ausgegeben werden.

Knoten Ein Knoten ist ein Teilnehmer im Bitcoin-Netzwerk, der die Bitcoin-Software laufen lässt und sich an die im Bitcoin-Protokoll definierten Regeln hält.

Miner Miner sind Teilnehmer im Bitcoin-Netzwerk, die neue Blöcke für die Blockchain erstellen.

Opcode Ist eine Zahl, die für eine bestimmte Operation steht, z. B. `0xAC = OP_CHECKSIG`.

Output Gibt an, an wen und wie viele Bitcoins gesendet werden.

RIPEMD-160 Eine des **R**esearch and Development in **A**dvanced **C**ommunications Technologies in **E**urope (RACE) Programms entwickelte kryptofische Hashfunktion.

SAT Satoshi ist die kleinste Einheit der Währung Bitcoin und nach dem Erfinder von Bitcoin Satoshi Nakamoto benannt. 100 000 000 SAT = 1 BTC.

Satoshi Nakamoto Erfinder der Kryptowährung Bitcoin.

secp256k1 Name der in Bitcoin verwendeten elliptischen Kurve.

SHA-256 Eine vom National Institute of Standards and Technology (NIST) entwickelte kryptofische Hashfunktion mit einer Hashwertgröße von 256 Bit.

Transaktion Eine Transaktion beschreibt den Besitzwechsel von Bitcoins.

Wallet Eine Wallet ist wie ein digitaler Geldbeutel, der die privaten Schlüssel eines Benutzers verwaltet und es ihm ermöglicht, Bitcoin zu empfangen, zu speichern und zu senden.

1 Einleitung

1.1 Einführung und Motivation

Bitcoin hat seinen Ursprung im Jahr 2009. Im Januar dieses Jahres wurde der erste Block der Bitcoin-Blockchain erzeugt. Dieser Block enthält eine versteckte Nachricht:

> The Times 03/Jan/2009 Chancellor on brink of second bailout for banks

Diese Nachricht bezieht sich auf einen Artikel in der *Times*, der die angespannte Lage während der Finanzkrise kommentiert. Zu dieser Zeit stand die britische Regierung kurz davor, einem Rettungspaket für Banken zuzustimmen. Der bis heute anonyme Erfinder von Bitcoin, Satoshi Nakamoto, zeigt sich in seiner Arbeit skeptisch gegenüber dem Bankensystem.

> „Banks must be trusted to hold our money and transfer it electronically, but they lend it out in waves of credit bubbles with barely a fraction in reserve. We have to trust them with our privacy, trust them not to let identity thieves drain our accounts. Their massive overhead costs make micropayments impossible."
>
> Satoshi Nakamoto, February 11, 2009

Nakamoto äußerte Kritik an Banken, die nicht nur als vertrauenswürdige Instanzen für die Verwaltung von Geld agieren müssen, sondern auch den Zugang zu Bankkonten verweigern und Konten einfrieren können. Die Notwendigkeit der Preisgabe persönlicher Daten zur Kontoführung sieht Nakamoto ebenfalls problematisch. Diese Punkte veranlassten ihn dazu, eine Alternative zu entwickeln.

> „The result is a distributed system with no single point of failure. Users hold the crypto keys to their own money and transact directly with each other, with the help of the P2P network to check for double-spending."
>
> Satoshi Nakamoto, February 11, 2009

Nakamoto analysierte bestehende Ansätze für digitales Geld und entwickelte eine dezentrale digitale Währung, die es jedem weltweit ermöglicht, Geld zu senden und zu empfangen – mit der einzigen Voraussetzung eines Internetzugangs. Es gibt hier keine zentrale Instanz, der Vertrauen entgegengebracht werden muss.
Bitcoin war die erste funktionierende Implementierung einer Kryptowährung in einem Peer-to-Peer-Netzwerk und bleibt bis heute die bekannteste und am weitesten verbreitete. Heute, 15 Jahre nach ihrer Einführung, hat Bitcoin kürzlich ein neues Allzeithoch erreicht. Die Relevanz dieser revolutionären digitalen Währung ist nach wie vor gegeben.
Die meisten sind sich der Existenz dieser digitalen Währung bewusst, verstehen jedoch oft nicht genau, wie sie funktioniert und warum sie möglicherweise ein neues Zeitalter

des Geldes eingeläutet hat. Um dies besser zu verstehen, ist es wichtig, die technischen Hintergründe von Bitcoin zu verstehen. Doch die technischen Grundlagen sind oft komplex und schwer verständlich. Die Literatur zu Bitcoin neigt dazu, entweder zu detailliert zu sein und weniger versierte Leser abzuschrecken oder so oberflächlich, dass wichtige technische Aspekte unbeschrieben bleiben.

1.2 Aufgabenstellung

Das Ziel dieser Forschungsarbeit ist es, die technischen Grundlagen von Bitcoin zu erklären und nachvollziehbar darzustellen.

2 Kryptografische Grundlagen

Jedes informationstechnologische System das sicher sein soll, muss zwangsläufig auf Kryptografie zurückgreifen, so auch Bitcoin. In diesem Kapitel werden die wichtigen kryptografischen Grundlagen von Bitcoin erläutert, darunter Hashfunktionen, Merkle Bäume, Schlüssel und digitale Signaturen.

2.1 Hashfunktionen

Hashfunktionen, auch als Streuwertfunktionen bezeichnet, spielen eine zentrale Rolle in nahezu allen Bereichen von Bitcoin. In diesem Abschnitt werden zunächst Hashfunktionen definiert und die damit verbundenen Begriffe erläutert. Die Größe der Zahl 2^{256} wird verdeutlicht, da sie entscheidend zur Sicherheit von Hashfunktionen beiträgt. Im weiteren Verlauf werden die grundlegenden Eigenschaften von Hashfunktionen vorgestellt. Dabei wird erläutert, welche Anforderungen erfüllt sein müssen, damit eine Hashfunktion als kryptografisch sicher gilt. Abschließend wird auf die in Bitcoin verwendeten Hashfunktionen eingegangen und ihre Anwendungsbereiche beschrieben.

2.1.1 Grundlagen

Eine Hashfunktion ist ein Algorithmus, der beliebige Eingabedaten (z. B. Dateien, Texte) in einen festen, meist kürzeren Ausgabewert umwandelt. Der Ausgabewert der Hashfunktion ist wie ein Fingerabdruck für die Eingabedaten.

$$\text{Hashfunktion}(\text{Datum}) = \text{Fingerabdruck}$$

Datum:

- kann beliebige Länge haben
- wird auch als Nachricht, Urbild oder im englischen als Preimage bezeichnet

Fingerabdruck:

- feste Länge, in Bitcoin 160 Bit, 256 Bit oder 512 Bit lang
- wird Hashwert, Hash, Streuwert oder im englischen Digest genannt

In Abbildung 2.1 ist der SHA-256-Hashwert der Nachricht „Hochschule München“ dargestellt. Hashwerte erscheinen auf Webseiten oder in Dokumentationen häufig als Hexadezimalzahl, um Platz zu sparen. Dabei sollte man sich bewusst sein, dass es 256 einzelne Bits (0 oder 1) sind, was insgesamt 2^{256} mögliche Hashwerte ergibt. Zur Verdeutlichung zeigt Abbildung 2.1 den Hashwert sowohl in binärer als auch in hexadezimaler Form.

Urbild:

Hochschule München

Hashwert (hexadezimal):

```
d5423bd3c7b7ac0fb0dc9eaf31b165120071bab151f142d1931e31cecda7d773
```

Hashwert (binär):

```
11010101010000100011101111010011110001111011011110101100000011111
10110000110111001001111010101111001100011011000101100101000100100
00000000011100011011101010110001010100011111000101000010110100001
10010011000111100011000111001110110011011010011111010111011100111
```

Abbildung 2.1: *Beispiel-Hash der SHA-256 Hashfunktion*

2.1.2 Verdeutlichung der Größe von 2^{256}

Da Hashfunktionen mit einer Hashwertlänge von 256 Bit in Bitcoin am häufigsten verwendet werden, wird hier exemplarisch die Größe von 2^{256} betrachtet.

$$2^{256} \approx 1,158 \times 10^{77}$$

Zum Vergleich: Die geschätzte Anzahl an Atomen im sichtbaren Universum beträgt etwa 10^{80} [1, S. 126].
Einer der aktuell schnellsten Supercomputer ist der Frontier Supercomputer mit einer Spitzenleistung von etwa 1,6 EF[1] [2]. Angenommen, ein sehr effizienter Algorithmus benötigt nur eine Operation, um einen Hashwert zu bestimmen[2]. Die benötigte Zeit, die der Frontier Supercomputer zum Erzeugen von 2^{256} Hashwerten benötigt, wird wie folgt berechnet:

Operationen pro Sekunde des Supercomputers: $1,6 \times 10^{18}$ FLOPS

Die benötigte Zeit t in Sekunden ist: $t = \dfrac{1,158 \times 10^{77}}{1,6 \times 10^{18} \frac{1}{s}} \approx 7,2 \times 10^{58}$ s

Ein Jahr hat etwa 31,5 Millionen Sekunden ($3,15 \times 10^{7}$ Sekunden). Die benötigte Zeit in Jahren beträgt:

$$t \approx \frac{7,2 \times 10^{58}}{3,15 \times 10^{7}} \approx 2,3 \times 10^{51} \text{ Jahre}$$

[1] EF = ExaFLOPS = $1,6 \times 10^{18}$ Floating Point Operations Per Second (FLOPS), also 1,6 Trillionen Fließkomma-Operationen pro Sekunde.
[2] in der Praxis deutlich mehr

Die Zeitspanne um 2^{256} Hashwerte zu berechnen übersteigt das geschätzte Alter des Universums von 13,7 Milliarden Jahren ($= 1,37 \times 10^{10}$ Jahre) [3] bei Weitem.

2.1.3 Eigenschaften

Hashfunktionen gibt es in zahlreichen Varianten, von denen einige mittlerweile nicht mehr verwendet werden, da sie als unsicher gelten oder die geforderten Eigenschaften nicht vollständig erfüllen. Je nach Anwendungsbereich müssen Hashfunktionen spezifische Anforderungen erfüllen, um ihre jeweilige Funktion zuverlässig zu gewährleisten. Im Folgenden werden zunächst die allgemeinen Eigenschaften von Hashfunktionen beschrieben, bevor auf die besonderen Merkmale eingegangen wird, die sichere kryptografische Hashfunktionen auszeichnen.

2.1.3.1 Grundlegende Eigenschaften

In diesem Unterabschnitt werden die Eigenschaften beschrieben, die jede Hashfunktion aufweisen muss, damit sie in der Praxis nutzbar ist.

Effizient Der Hashwert muss schnell und mit geringem Speicherverbrauch berechnet werden können.

Deterministisch Die Hashfunktion muss bei gleicher Eingabe immer denselben Hashwert zurückgeben.

Feste Länge Der Hashwert muss eine feste Länge haben.

Gleichverteilung Jeder mögliche Hashwert soll gleich wahrscheinlich sein, sodass die Hashwerte gleichmäßig auf die möglichen Eingabewerte verteilt sind.

Lawineneffekt Eine kleine Änderung des Urbilds bewirkt eine große Änderung des Hashwerts[3].

In Abbildung 2.2 ändert sich etwa die Hälfte der Bits des Hashwerts durch die kleine Änderung von „ü“ zu „u“ im Urbild.

[3] Etwa die Hälfte der Bits ändern sich.

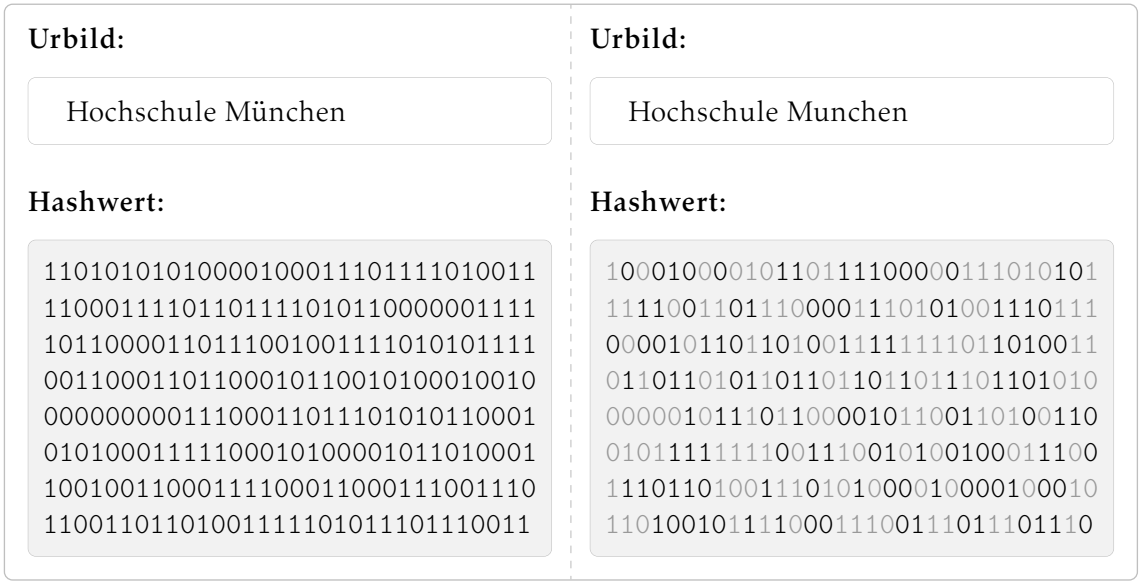


Abbildung 2.2: *Lawineneffekt*

2.1.3.2 Eigenschaften kryptografischer Hashfunktionen

Eine kryptografische Hashfunktion ist eine Hashfunktion, die so konzipiert ist, dass sie spezielle Eigenschaften wie Kollisionsresistenz und Urbildresistenz aufweist, die für viele Anwendungen in der Informationssicherheit wichtig sind [4, S. 4]. Eine Hashfunktion wird als sicher betrachtet, wenn sie die folgenden Eigenschaften erfüllt.

Urbildresistenz

Diese Eigenschaft stellt sicher, dass es rechnerisch unmöglich ist, aus einem gegebenen Hashwert die ursprüngliche Nachricht zu bestimmen. Kryptografische Hashfunktionen sind aufgrund dieser Eigenschaft Einwegfunktionen. [5, S. 339]

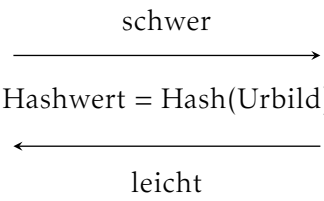


Abbildung 2.3: *Veranschaulichung Urbildresistenz*

Einschub Kollision Eine Kollision tritt auf, wenn zwei unterschiedliche Nachrichten denselben Hashwert erzeugen. Aufgrund der endlichen Anzahl möglicher Hashwerte und der prinzipiell unbegrenzten Anzahl möglicher Eingaben ist dies unvermeidlich. Hashfunktionen sollen jedoch so gestaltet sein, dass es praktisch unmöglich ist, Kollisionen

absichtlich herbeizuführen.

Schwache Kollisionsresistenz (zweite Urbildresistenz)

Es soll praktisch unmöglich sein, eine Nachricht x_2 zu konstruieren, deren Hashwert mit einem vorgegebenen Hashwert übereinstimmt. Die einzige Möglichkeit einen bestimmten Hashwert zu erhalten, besteht im systematischen Ausprobieren (Brute-Force-Methode), bis der gewünschte Wert erreicht wird. Da jeder Hashwert mit gleicher Wahrscheinlichkeit auftritt, wäre eine extrem lange Suche erforderlich (siehe Abschnitt 2.1.2).

$$\text{hash}(x_1) \;=\; \text{hash}(x_2 = ?) \qquad \text{mit} \quad x_1 \neq x_2 \quad [6, \text{S. } 339]$$

Abbildung 2.4: *Veranschaulichung der schwachen Kollisionsresistenz*

Starke Kollisionsresistenz

Die starke Kollisionsresistenz ist die allgemeinere Form der schwachen Kollisionsresistenz. Sie garantiert, dass es praktisch unmöglich ist, zwei beliebige unterschiedliche Nachrichten x_1 und x_2 zu finden, die denselben Hashwert erzeugen.

$$\text{hash}(x_1 = ?) \;=\; \text{hash}(x_2 = ?) \qquad \text{mit } x_1 \neq x_2 \quad [6, \text{S. } 339]$$

Wird bei der starken Kollisionsresistenz eine Nachricht $x1$ vorgegeben, so erhält man die schwache Kollisionsresistenz. Die starke Kollisionsresistenz ist schwerer zu erreichen und meistens die Eigenschaft, die als erstes gebrochen wird. In solchen Fällen gilt die Hashfunktion als unsicher.

2.1.4 Anwendungsmöglichkeiten

Die beschriebenen Eigenschaften von Hashfunktionen eröffnen zahlreiche Anwendungsmöglichkeiten, die für Bitcoin essenziell sind.

2.1.4.1 Datenintegrität

Um die Integrität von Daten sicherzustellen, wird ein Hashwert für die ursprünglichen Daten erzeugt und gespeichert. Soll überprüft werden, ob die Daten manipuliert wurden,

wird der Hashwert der aktuellen Daten berechnet. Weicht dieser von dem ursprünglichen Hashwert ab, so wurden die Daten verändert. Über den Lawineneffekt werden kleine Änderungen in den Daten sofort im Hashwert der Nachricht erkennbar. Die Kollisionsresistenz gewährleistet, dass kein zweiter Datensatz erstellt werden kann, der einen gleichen Hashwert erzeugt.

2.1.4.2 Hash-Pointer

Der Hashwert, der im Abschnitt 2.1.4.1 für die Daten erstellt wurde, kann an einer anderen Stelle gespeichert werden und als Hash-Pointer dienen. Über diesen Hash-Pointer kann auf die referenzierten Daten zugegriffen werden. Aufgrund der Kollisionsresistenz ist ein Hash-Pointer eindeutig und kann nicht auf zwei Datensätze gleichzeitig verweisen.

2.1.4.3 Commitment-Verfahren

Eine weitere Anwendung ist das sogenannte Commitment-Verfahren. Hierbei wird zunächst der Hashwert veröffentlicht. Zu einem späteren Zeitpunkt kann das zugehörige Urbild präsentiert werden. Es lässt sich dadurch beweisen, dass man zum Zeitpunkt der Veröffentlichung des Hashwerts im Besitz des Urbildes war, ohne dieses vorab offenzulegen.

2.1.5 Hashfunktionen in Bitcoin

Es gibt eine Vielzahl unterschiedlicher Hashfunktionen. In den folgenden Unterabschnitten werden die Hashfunktionen vorgestellt, die in Bitcoin verwendet werden.

2.1.5.1 SHA-256

Der Secure Hash Standard (SHS) beschreibt verschiedene Secure Hash Algorithms (SHAs), die vom NIST standardisiert wurden. Darin wird auch die oft verwendete Hashfunktion SHA-256 beschrieben.

Nachrichtengröße	$< 2^{64}$ Bit $\approx$ 2.300.000 Terabyte
Größe Hashwert	256 Bit

Tabelle 2.1: *Eigenschaften der SHA-256 Hashfunktion [7]*

Die mögliche Nachrichtengröße aller in Bitcoin verwendeten Hashfunktionen ist so groß, dass diese nie überschritten werden kann. Sie wird für die weiteren Hashfunktionen nicht mehr separat angegeben. Die SHA-256 Hashfunktion findet mittlerweile in Bitcoin bei dem Standart-Skript Pay To Witness Script Hash (P2WSH) Einsatz (siehe Abschnitt 4.5), wird jedoch nicht so häufig verwendet wie HASH256.

2.1.5.2 HASH256

In Bitcoin werden Hashwerte fast immer über die HASH256 Hashfunktion berechnet. Dabei wird die SHA-256-Hashfunktion zunächst einmal auf die Eingabedaten angewendet, und der resultierende Hashwert wird erneut mit SHA-256 verarbeitet:

HASH256(Nachricht) = SHA-256(SHA-256(Nachricht))

In Bitcoin wird primär diese Hashfunktion verwendet. An allen Stellen, an denen die folgenden Begrifflichkeiten in dieser Arbeit verwendet werden, wurden die Daten mit der HASH256-Hashfunktion verarbeitet:

- Hashwert der Transaktionsdaten (wird als Transaktions ID (TxID) bezeichnet)
- Block Hashwert
- Merkle Baum
- Prüfsummen
- Hashwerte für Signaturen

Die HASH256 Hashfunktion besitzt die gleichen Eigenschaften wie die SHA-256 Hashfunktion, da beide denselben Algorithmus verwenden (siehe Tabelle 2.1).

2.1.5.3 HASH160

Die HASH160-Hashfunktion kombiniert die beiden Hashfunktionen SHA-256 und RIPEMD-160.

HASH160(Nachricht) = RIPEMD-160(SHA-256(Nachricht))

Die RIPEMD-160-Hashfunktion erzeugt einen Hashwert mit einer Länge von 160 Bit, wodurch auch der von HASH160 generierte Hashwert 160 Bit lang ist [8, S. 4]. HASH160 wird zum Kürzen eines öffentlichen Schlüssels oder Skripts vor der Umwandlung in eine Adresse verwendet.

> „Im Gegensatz zu SHA-1 und SHA-2 ist er der einzige Algorithmus, der nicht von NIST und NSA entwickelt wurde, sondern von einer Gruppe europäischer Wissenschaftler. Obwohl es keinerlei Anzeichen gibt, dass die SHA-Algorithmen künstliche Schwachstellen enthalten (die beispielsweise von US-Nachrichtendiensten bewusst eingebaut wurden), kann RIPEMD-160 attraktiv für Personen sein, die allgemein kein Vertrauen in Regierungsalgorithmen haben.“ [5, S. 357]

Es könnte sein, dass Satoshi Nakamoto sich für RIPEMD-160 entschieden hat und gegen SHA-1, welche ebenso eine Hashwertgröße von 160 Bit hat, um eine Hashfunktion zu verwenden, die nicht von der amerikanischen Regierung entwickelt wurde. Das ist aber reine Spekulation, da sich Satoshi Nakamoto nicht dazu geäußert hat.

2.1.5.4 Weitere Hashfunktionen

Zusätzlich zu den zuvor beschriebenen Hashfunktionen werden in Bitcoin zwei weitere Hashfunktionen verwendet, die in dieser Arbeit lediglich der Vollständigkeit halber erwähnt werden, da ihr Anwendungsbereich zu spezifisch für diese Arbeit ist.

- HMAC-SHA512: Diese Hashfunktion wird in Hierarchical Deterministic Wallets (HD Wallets) genutzt, um alle Schlüssel aus einer einzigen Quelle abzuleiten.
- Password Based Key Derivation Function 2 (PBKDF2): Diese Funktion erzeugt über wiederholtes Anwenden von HMAC-SHA512 eine sichere Quelle für die Schlüsselableitung aus einer benutzerfreundlichen Passphrase.

2.2 Hash-Baum (Merkle Baum)

Der erste Hash-Baum wurde von Ralph Merkle beschrieben [9]. Hash-Bäume werden deshalb auch Merkle Bäume genannt.
Ein Hash-Baum ist ein binärer Baum, dessen Blätter die Hashwerte der Daten sind. In Bitcoin sind das die TxIDs. Der Hashwert eines Elternknotens wird erzeugt, indem die beiden Kinder miteinander verknüpft und deren Hashwert berechnet wird. Der Hashwert eines beliebigen Knotens im Baum lässt sich rekursiv über die folgende Formel bestimmen:

$$\text{Knoten}_i = \text{Hash}(\text{Knoten}_{2i} + \text{Knoten}_{2i+1})$$

Ist die Anzahl der Knoten auf einer Ebene des Baums ungerade, so können nicht alle Hashwerte in Zweiergruppen aufgeteilt werden. Es muss der letzte Knoten dieser Ebene mit sich selbst verknüpft werden, um den Hashwert des Elternknotens berechnen zu können. Dies kann im Knoten H_{33} in Abbildung 2.5 nachvollzogen werden.
Der Hashwert eines Elternknotens ist ein Hash-Pointer, der auf die beiden Kinder verweist[4]. Wie in Abschnitt 2.1.4.1 und Abschnitt 2.1.4.2 erläutert, gewährleisten Hash-Pointer die Integrität der Daten, auf die sie zeigen. Betrachtet man den Baum als Graphen und die Hash-Pointer als Relation „sichert Integrität von“, so ist diese Relation transitiv. Das bedeutet, dass die Merkle-Root die Integrität aller im Baum enthaltenen Daten gewährleistet.

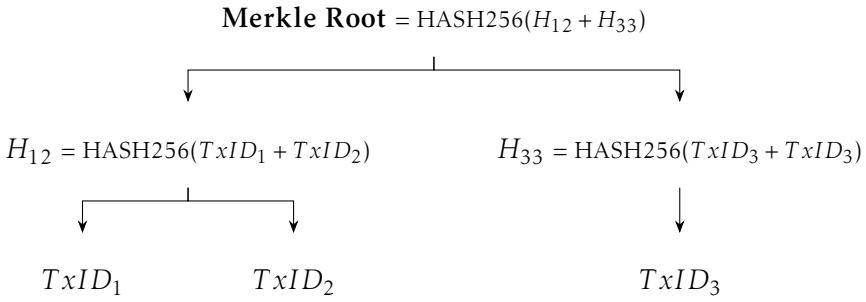


Abbildung 2.5: *Merkle Baum mit 3 Blättern*

Es existieren alternative Ansätze zum Hash-Baum, um die Integrität aller Transaktionen zu gewährleisten. Beispielsweise könnte ein Hashwert der konkatenierten TxIDs erstellt werden. Bei diesem Ansatz müssen alle Transaktionen bekannt sein, um die Zugehörigkeit einer Transaktion in der Menge zu prüfen.
Ein Hash-Baum bietet den Vorteil, dass zum Verifizieren einer Transaktion nur die Transaktion selbst und der zugehörige Merkle Proof benötigt werden. Der Merkle Proof sind alle Hashwerte die auf dem Pfad zur Merkle Root noch fehlen und sind in der Abbildung 2.6 blau markiert. Abbildung 2.6 zeigt wie geprüft wird, ob die Transaktion Tx_5 im Hash-Baum enthalten ist. [10]

[4] Der Hash-Pointer zeigt nicht aufgrund von Kollisionen auf 2 Kinderknoten, sondern weil die Hashwerte dieser Knoten konkateniert wurden.

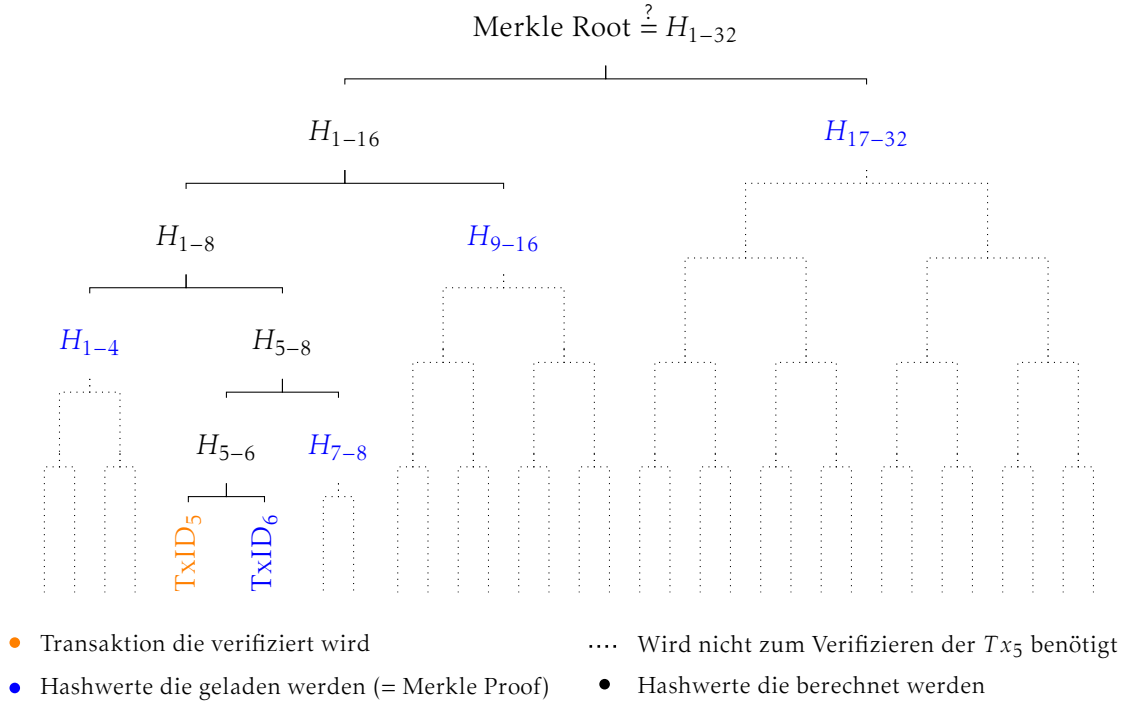


Abbildung 2.6: *Transaktion Tx_5 über Merkle Proof verifizieren*

Ein Hash-Baum beinhaltet in Bitcoin üblicherweise mehrere tausend Transaktionen. Die Anzahl an Hashwerten des Merkle Proofs kann über die Höhe des Baums bestimmt werden.

$$\text{Anzahl Hashwerte} = \text{Höhe des Baums} - 1$$

Die Baumhöhe in Bitcoin entspricht:

$$\text{Höhe des Baums} = \lceil log_2(\text{Anzahl Transaktionen}) \rceil + 1$$

Sind beispielsweise 2500 Transaktionen im Hash-Baum, so müssten lediglich 12 Hashwerte anstelle von 2500 geladen werden, was den Speicherbedarf deutlich reduziert.

$$\lceil log_2(2500) \rceil = 12$$

2.3 Schlüssel und Signaturen

Die in diesem Abschnitt dargestellten Grundlagen beziehen sich direkt auf Bitcoin. Es werden beispielsweise keine verschiedenen Schlüsseltypen erläutert, sondern ausschließlich die in Bitcoin verwendeten.
Bitcoin verwendet Schlüssel, die auf elliptischen Kurven basieren. Zunächst wird die spezifische elliptische Kurve und ihre Parameter vorgestellt, die in Bitcoin zum Einsatz kommt. Im Anschluss werden die mathematischen Operationen beschrieben, die auf dieser elliptischen Kurve ausgeführt werden können. Mit diesen Operationen wird schließlich der Elliptic Curve Digital Signature Algorithm (ECDSA) erklärt, der für die Generierung der Schlüssel sowie das Erstellen und Verifizieren von Signaturen verwendet wird.
Die Inhalte dieses Abschnitts basieren auf den Spezifikationen und Informationen aus den beiden Quellen „SEC 1: Elliptic Curve Cryptography“ [11] und „FIPS PUB 186-5: Digital Signature Standard (DSS)“ [12]. Diese dienen als Grundlage für die folgenden Unterabschnitte.

2.3.1 Elliptische Kurve secp256k1

Elliptische Kurven für die elliptische Kurven Kryptographie (ECC) können auf verschiedenen Typen endlicher Felder definiert werden. Diese endlichen Felder gewährleisten sowohl die theoretische Sicherheit der ECC als auch die praktische Umsetzung der Algorithmen[5], die elliptische Kurven auf diesen Feldern verwenden. Die in Bitcoin genutzte elliptische Kurve wird auf dem endlichen Primfeld $\mathbb{F}_p$ definiert. Kurven die auf diesem Primfeld definiert sind haben folgende allgemeine Form:

$$E : y^2 \equiv x^3 + ax + b \mod p \qquad \text{mit a, b} \in \mathbb{F}_p$$

Satoshi Nakamoto wählte für Bitcoin die elliptische Kurve secp256k1, die durch die folgenden Parameter definiert ist:

[5] Beispielsweise ECDSA

p	FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F $= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$
a	0
b	7
G	x: 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 y: 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8
n	FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141

Tabelle 2.2: *Parameter secp256k1 [13, S. 9]*

Diese Parameter haben folgende Bedeutung:

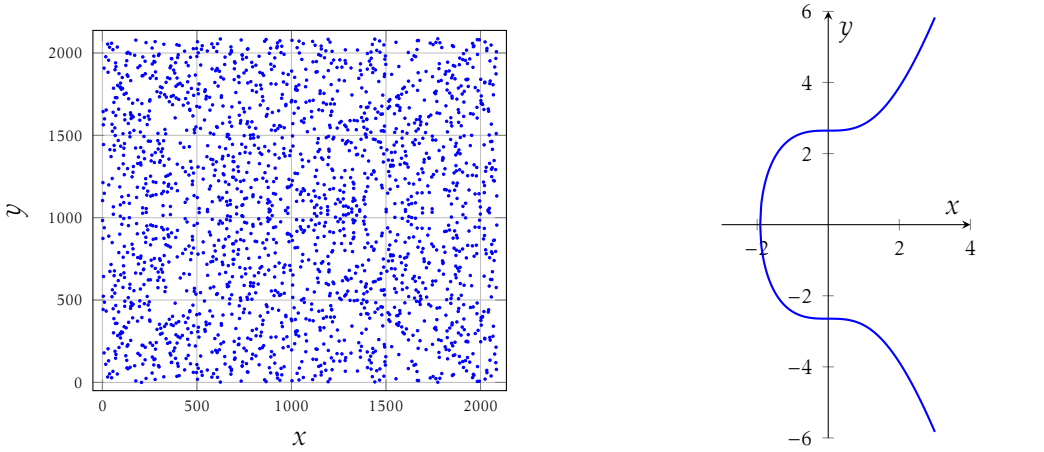
p: Eine große Primzahl, die den Bereich der Werte für x und y beschränkt, die auf der Kurve liegen können.

a, b: Die Koeffizienten der elliptischen Kurvengleichung $\Rightarrow E : y^2 = x^3 + 7$

G: Der Generatorpunkt (Basispunkt) der Kurve. Er wird durch seine x- und y-Koordinaten angegeben. Der Punkt G ist der Ausgangspunkt für die mathematischen Berechnungen auf der Kurve.

n: Die Ordnung des Punktes G. Sie gibt an, wie viele Punkte der Kurve erreicht werden können mit $n < p$.

Die in der Abbildung 2.7a gezeigte Kurve wurde auf einem kleineren endlichen Feld abgebildet. So ähnlich würde der Plot zu secp256k1 aussehen.
Zur grafischen Veranschaulichung der Operationen auf elliptischen Kurven werden in den folgenden Abschnitten jedoch Plots im Reellen verwendet, wie beispielsweise in Abbildung 2.7b. Diese Darstellung erleichtert das Verständnis der geometrischen Zusammenhänge. Die mathematischen Operationen funktionieren aber in beiden Fällen gleich.



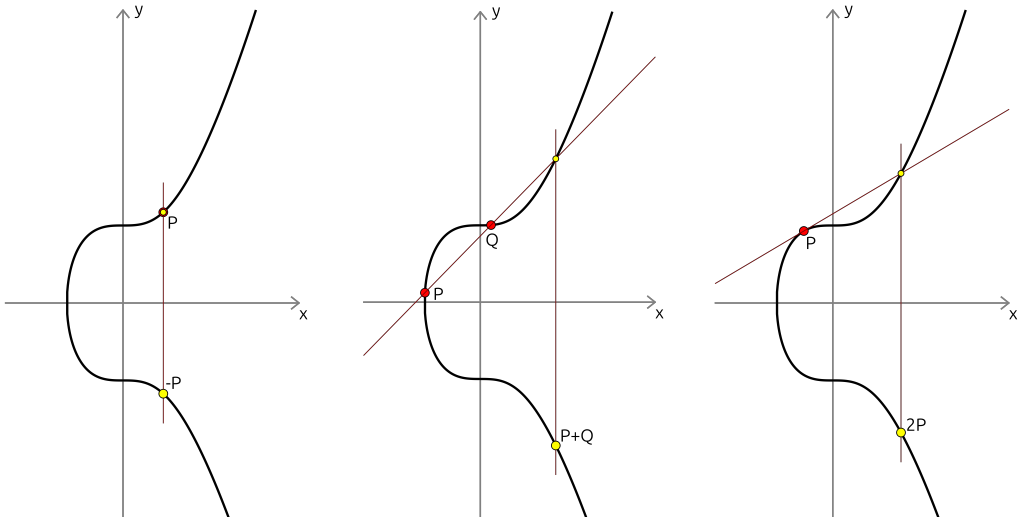
(**a**) *secp256k1 über dem endlichen Primfeld* $\mathbb{F}_{2087}$*:* $y^2 = x^3 + 7 \mod 2087$

(**b**) *secp256k1 im Reellen:* $y^2 = x^3 + 7$

Abbildung 2.7: *Veranschaulichung der elliptischen Kurven secp256k1*

2.3.2 Punkt-Operationen auf secp256k1

Auf elliptischen Kurven gibt es die beiden grundlegenden Punkt-Operationen **Addition** und **Verdopplung**. Diese Operationen bilden die Grundlage der elliptischen Kurvenarithmetik. Über wiederholte Anwendung dieser Operationen kann ein Punkt effizient mit einem Skalar multipliziert werden.



(**a**) *Der negative Punkt* $(-P)$ *von* P

(**b**) $P + Q$

(**c**) $P + P = 2P$

Abbildung 2.8: *Punkt-Operationen auf elliptischen Kurven. Grafiken wurden mit dem Elliptic Curve Plotter erstellt [14]*

2.3.2.1 Addition

Zwei Punkte werden addiert (siehe Abbildung 2.8b), indem eine Gerade durch beide Punkte gelegt wird und der Punkt, der die Kurve schneidet negiert wird (siehe Abbildung 2.8a).
Mathematisch wird die Summe der Punkte (x_1, y_1) und (x_2, y_2) wie folgt berechnet:

$$\begin{array}{c} (x_1,y_1)+(x_2,y_2)=(x_3,y_3) \quad \text{mit:} \\ \lambda \equiv \frac{y_2-y_1}{x_2-x_1} \mod p \quad [6], \quad x_3 \equiv \lambda^2 - x_1 - x_2 \mod p, \quad y_3 \equiv \lambda(x_1-x_3)-y_1 \mod p \end{array}$$

2.3.2.2 Verdoppelung

Es wird ein Punkt geometrisch zu sich selbst addiert (siehe Abbildung 2.8c), indem eine Tangente an den Punkt gelegt wird und der Punkt an dem die Tangente die Kurve schneidet negiert wird Abbildung 2.8a).
Der verdoppelte Punkt kann wie folgt berechnet werden:

$$\begin{array}{c} (x_1,y_1)+(x_1,y_1)=(x_3,y_3) \quad \text{mit:} \\ \lambda \equiv \frac{3x_1^2+a}{2y_1} \mod p \ [7], \quad x_3 \equiv \lambda^2 - 2x_1 \mod p, \quad y_3 \equiv \lambda(x_1-x_3)-y_1 \mod p \end{array}$$

2.3.2.3 Skalare Multiplikation

Die skalare Multiplikation ist das Herzstück der ECC. Ein Punkt P auf einer elliptischen Kurve kann mit einem Skalar k multipliziert werden, indem der Punkt k-mal zu sich selbst addiert wird. Die Berechnung von $k*P$ kann effizient über die Addition und Verdoppelung berechnet werden.
Ein Beispiel zur Bestimmung von $26P$ ist in [5, S. 283] beschrieben:

$26_{10} = 11010_2$
Das erstes Bit wird ignoriert. Für jede „0“ eine Verdoppelung und für jede „1“ eine Verdoppelung und Addition durchführen:

$$1P \xrightarrow{\text{verdoppeln}} 2P \xrightarrow{\text{addieren}} 3P \xrightarrow{\text{verdoppeln}} 6P \xrightarrow{\text{verdoppeln}} 12P \xrightarrow{\text{addieren}} 13P \xrightarrow{\text{verdoppeln}} 26P$$

Durch diese Methode wird die Anzahl der benötigten Operationen erheblich reduziert. Statt 25 einfacher Additionen werden nur vier Verdoppelungen und zwei Additionen ausgeführt. Dies führt bei großen Zahlen wie 2^{256} zu einer deutlichen Effizienzsteigerung. Die maximale Anzahl an Operationen bei der skalaren Multiplikation auf secp256k1 ergibt sich aus der Anzahl der Stellen der Binärzahl plus der Anzahl von „1“ in der

[6] λ entspricht hier der Steigung der Geraden die in Abbildung 2.8b dargestellt ist. In der modularen Arithmetik erfolgt die Division durch Multiplikation mit dem modularen Inversen:
$\lambda \equiv (y_2 - y_1) * (x_2 - x_1)^{-1} \mod p$

[7] Ist die Tangentensteigung in Abbildung 2.8b und muss über das multiplikative Inverse berechnet werden:
$\lambda \equiv (3x_1^2 + a) * (2y_1)^{-1} \mod p$

Binärdarstellung des Skalars. Da mit 256-Bit großen Zahlen gearbeitet wird, sind maximal 512 Operationen erforderlich.
Die skalare Multiplikation ist eine Einwegfunktion: Während $kG = P$ effizient berechnet werden kann, ist es praktisch unmöglich, aus P das Skalar k zu ermitteln. Die einzige Möglichkeit ist, den Generatorpunkt G wiederholt zu sich selbst zu addieren, bis der Punkt P erreicht wird. Dies ist jedoch aufgrund der enormen Größe des Wertebereichs ($\approx 2^{256}$, siehe Tabelle 2.2 und Abschnitt 2.1.2) in endlicher Zeit nicht möglich.

2.3.3 Elliptic Curve Digital Signature Algorithm (ECDSA)

Der ECDSA, ein von der American National Standards Institute (ANSI) standardisierter Algorithmus, wird in Bitcoin eingesetzt, um den Besitz von Bitcoins kryptografisch abzusichern. Mit dem ECDSA lassen sich Schlüssel erzeugen, die sowohl zur Erstellung als auch zur Verifizierung von digitalen Signaturen dienen.

2.3.3.1 Schlüsselgenerierung

Privater Schlüssel k_{pr} Der private Schlüssel ist vergleichbar mit einem Passwort wie man es aus dem Bankensystem kennt. Er ermöglicht den Zugriff auf das Bitcoin-Guthaben und ist daher von zentraler Bedeutung. Der private Schlüssel muss unbedingt geheim gehalten und darf nicht weitergegeben werden. Der Verlust des privaten Schlüssels bedeutet unwiderruflich den Verlust der zugehörigen Bitcoins. Gelangt der Schlüssel in die Hände Unbefugter, können diese das Guthaben stehlen.
Private Schlüssel können von jedem Nutzer einfach selbst generiert werden. Technisch gesehen ist ein privater Schlüssel eine zufällig gewählte Zahl im Bereich von 1 bis $n \approx 2^{256}$. Aufgrund dieses enorm großen Zahlenbereichs ist die Wahrscheinlichkeit, dass zwei Personen zufällig denselben privaten Schlüssel wählen, praktisch gleich null. Entscheidend ist jedoch die Verwendung eines zuverlässigen Zufallszahlengenerators bei der Generierung. Nur so kann gewährleistet werden, dass die Zahl tatsächlich zufällig ist. Ein schlechter oder vorhersehbarer Zufallszahlengenerator könnte dazu führen, dass private Schlüssel mehrfach verwendet oder vorhersagbar werden, was die Sicherheit der zugehörigen Bitcoins erheblich gefährden würde.

Öffentlicher Schlüssel K_{pub} Der öffentliche Schlüssel K_{pub} kann mit einer Kontonummer verglichen werden. Er kann aus dem privaten Schlüssel k_{pr} über die elliptische Kurve secp256k1 bestimmt werden. Dabei wird der Generatorpunkt G skalar mit dem privaten Schlüssel k_{pr} multipliziert.

$$K_{pub} = k_{pr}G$$

Da die skalare Multiplikation eine Einwegfunktion ist, ist es rechnerisch praktisch unmöglich aus dem öffentlichen Schlüssel den privaten zu berechnen (siehe Abschnitt 2.3.2.3).

2.3.3.2 Digitale Signaturen

Digitale Signaturen dienen als Nachweis, dass eine Person im Besitz des privaten Schlüssels zu einem bestimmten öffentlichen Schlüssel ist. Signaturen sind, im Gegensatz zu privaten Schlüsseln, nur für eine spezifische Nachricht gültig. In Bitcoin gewährleisten sie die Authentizität und Integrität einer Transaktion, indem sie belegen, dass die Transaktion von dem Besitzer des zugehörigen privaten Schlüssels autorisiert wurde und dass die Transaktionsdaten seit ihrer Signierung unverändert geblieben sind.
Eine Signatur wird erstellt, indem eine Nachricht[8] mit dem privaten Schlüssel signiert wird. Da der öffentliche Schlüssel aus dem privaten Schlüssel generiert wird, besteht eine eindeutige mathematische Verbindung zwischen der Signatur und dem öffentlichen Schlüssel. Diese Verbindung ermöglicht es, über die Signatur nachzuweisen, dass eine Person im Besitz des privaten Schlüssels zu einem öffentlichen Schlüssel ist, ohne den privaten Schlüssel selbst offenlegen zu müssen.
Abbildung 2.9 veranschaulicht den Prozess der Signaturerstellung und -verifizierung, der in den folgenden Unterabschnitten erläutert wird.

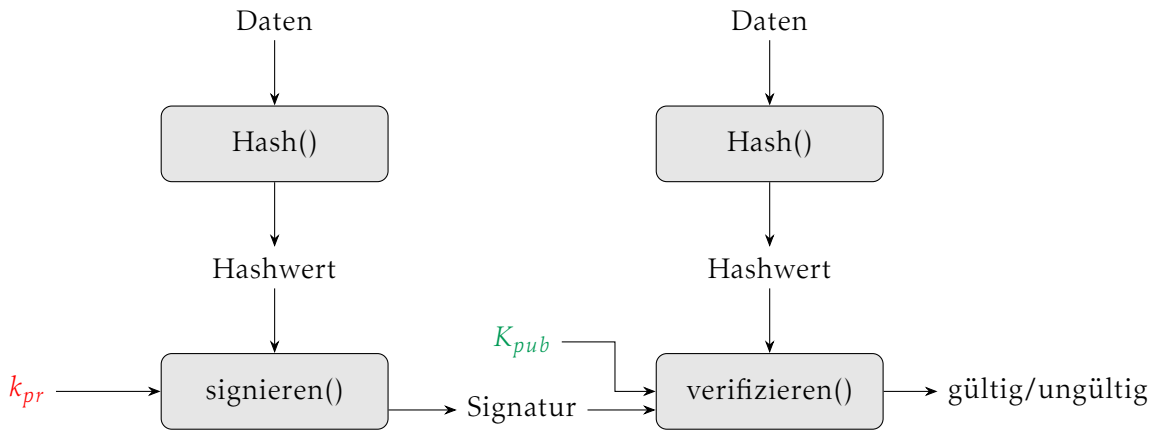


Abbildung 2.9: *Digitaler Signatur-Prozess. Veränderte Darstellung nach FIPS 186-5 [12, S. 9]*

Digitale Signaturen erstellen

$$\text{signieren}(\text{Nachricht}, k_{pr}) = \text{Signatur}$$

Die ECDSA Signatur $S = (r, s)$ der Nachricht M besteht aus den beiden Komponenten r und s. Als Input bekommt der Algorithmus den privaten Schlüssel k_{pr} und eine Nachricht M, sowie die Parameter der verwendeten elliptischen Kurve secp256k1.
Zunächst wird ein temporärer Schlüssel k über einen Random Number Generator (RNG) generiert.

$$k = \texttt{RNG(\{1, 2, ..., n - 1\})}$$

[8] In Bitcoin sind dies die Transaktionsdaten

Dadurch, dass k zufällig gewählt ist, wird beim mehrmaligen Signieren derselben Nachricht eine unterschiedliche Signatur erzeugt.

$$R^{sign} = kG = (R_x^{sign},\ R_y^{sign})$$
$$r \equiv R_x^{sign} \mod n$$

$$s \equiv k^{-1}(rk_{pr} + \texttt{HASH256}(M)) \mod n$$

Digitale Signaturen verifizieren Zum Verifizieren wird die Nachricht selbst, die Signatur der Nachricht und der öffentliche Schlüssel des privaten Schlüssels, mit dem die Signatur erstellt wurde, der verifizieren-Methode übergeben. Diese gibt wahr oder falsch zurück, je nachdem, ob die Signatur gültig ist oder nicht.

$$\text{verifizieren(Signatur, Nachricht, } K_{pub}) = \text{gültig/ungültig}$$

Mathematisch betrachtet wird der Generatorpunkt skalar mit dem Produkt aus s^{-1} und dem Hash der Nachricht $\texttt{SHA-256}(M)$ multipliziert. Der öffentliche Schlüssel wird skalar mit $s^{-1} \cdot r$ multipliziert.

$$\underbrace{(s^{-1} \cdot \texttt{HASH256}(M))}_{\text{Skalar}} \cdot \underbrace{G}_{\text{Punkt}} + \underbrace{(s^{-1}r)}_{\text{Skalar}} \cdot \underbrace{K_{pub}}_{\text{Punkt}} = \underbrace{R^{verify}}_{\text{Punkt}} \stackrel{?}{=} R^{sign}$$

Ist die Signatur valide, so ist die Addition der eben berechneten Punkte gleich dem Punkt R^{sign}. Zur Erinnerung: Die Signatur besteht aus den beiden Komponenten r und s, wobei $r \equiv R_x^{sign} \mod n$.

$$R_x^{verify} \begin{cases} \equiv R_x^{sign} & \mod n \equiv r \quad \Rightarrow \text{ gültige Signatur} \\ \not\equiv R_x^{sign} & \mod n \equiv r \quad \Rightarrow \text{ ungültige Signatur} \end{cases}$$

3 Transaktionen

Dieses Kapitel gibt einen umfassenden Überblick über Bitcoin-Transaktionen. Zunächst wird der Aufbau einer Transaktion beschrieben. Anschließend wird erläutert, wie das Problem der doppelten Ausgaben (Double-Spending) durch die Verwaltung der Menge aller Unspent Transaction Outputs (UTxOs) effektiv vermieden wird. Abschließend wird das Zusammenspiel mehrerer Transaktionen anhand eines fiktiven Ausschnitts aus einer Transaktionshistorie veranschaulicht.

3.1 Grundlegender Aufbau einer Transaktion

Um die Funktionsweise von Transaktionen besser zu verstehen, werden diese auf ihre zentralen Bestandteile reduziert. Eine Transaktion setzt sich zwingend aus den folgenden Bestandteilen zusammen.

- **Referenzierte Bitcoins**: Gibt an, welche Bitcoins gesendet werden sollen.
- **Autorisierung durch den Absender**: Die Transaktion muss über die Signatur des Absenders autorisiert werden.
- **Übertragene Bitcoin-Menge**: Gibt an, welcher Betrag in der Transaktion transferiert wird.
- **Empfänger**: Gibt an, an wen die Bitcoins gesendet werden und wie sie später wieder ausgegeben werden können.

In Abbildung 3.1 ist eine beispielhafte Transaktion mit der TxID $28a...cb2$ dargestellt. Die TxID ist der HASH256-Hashwert der Transaktionsdaten. Diese Transaktion verfügt über zwei Eingänge (Inputs) und zwei Ausgänge (Outputs).

3.1.1 Output

Ein Output definiert, wie viele Bitcoins an welchen Empfänger gesendet werden. Durch mehrere Outputs können Beträge an verschiedene Empfänger aufgeteilt werden. Ein Output kann auch genutzt werden, um sich selbst Rückgeld zu schicken, falls die Inputs den benötigten Betrag übersteigen [15, S. 5].
Ein Output lässt sich mit einer Box vergleichen, in der eine bestimmte Menge an Bitcoins verschlossen ist. Die genaue Menge wird im Transaktionsfeld `Value` angegeben. Der `ScriptPubKey` fungiert als das Schloss dieser Box. Solange ein Output in keiner weiteren Transaktion ausgeben wurde, wird er als UTxO bezeichnet.

3.1.2 Input

Ein Input verweist über den Hash-Pointer `Previous TxID` auf eine vorherige Transaktion. Mithilfe des `Index` wird ein bestimmter UTxO innerhalb der referenzierten Transaktion ausgewählt. Da die referenzierte Box mit Bitcoins verschlossen ist, muss im Input ein

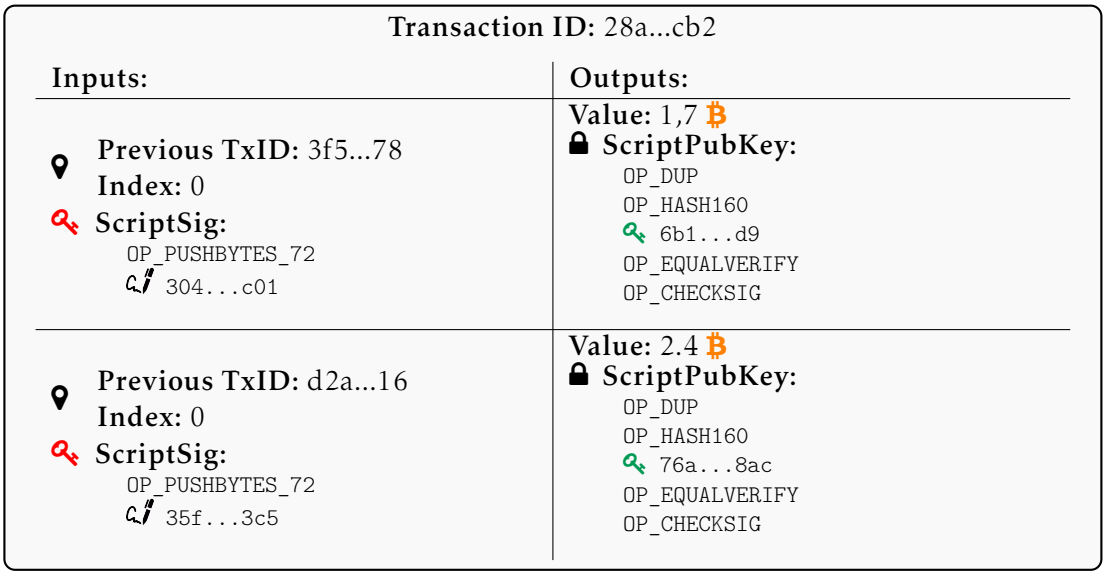


Abbildung 3.1: *Vereinfachter Aufbau einer Transaktion*

Schlüssel angegeben werden, mit dem sie entsperrt werden kann. Das `ScriptSig` entspricht diesem Schlüssel und dient als Nachweis dafür, dass der Absender im Besitz der referenzierten Bitcoins ist.
Es können mehrere Inputs kombiniert werden, um einen gewünschten Betrag zu erreichen, der ausgegeben werden soll [15, S. 5].

3.2 UTxO-Menge

In jeder neuen Transaktion muss geprüft werden, ob der referenzierte Output bereits ausgegeben wurde, um doppelte Ausgaben zu vermeiden. In Bitcoin wird dies dadurch sichergestellt, dass in allen vorherigen Transaktionen überprüft wird, ob der Output bereits in einem anderen Input verwendet wurde. Da dieser Vorgang aufwendig ist, speichert jeder Teilnehmer des Netzwerks die Menge aller UTxOs. Dadurch wird der Suchaufwand erheblich reduziert. Die UTxO-Menge wird nach jeder ausgeführten Transaktion aktualisiert und umfasst alle Bitcoins, die aktuell im Umlauf sind.
Der Besitz von Bitcoins ergibt sich aus der Fähigkeit, UTxOs zu entsperren. Der eigene Kontostand berechnet sich aus der Summe aller UTxOs, für die man im Besitz des passenden Schlüssels ist.

3.3 Transaktionen im Überblick

Abbildung 3.2 soll einen Überblick verschaffen, wie Transaktionen in Bitcoin ablaufen. Eine Transaktion stellt einen Knoten im Graphen dar. Die Kanten repräsentieren den Bitcoinfluss, während die Gewichte die Menge der transferierten Bitcoins angeben.

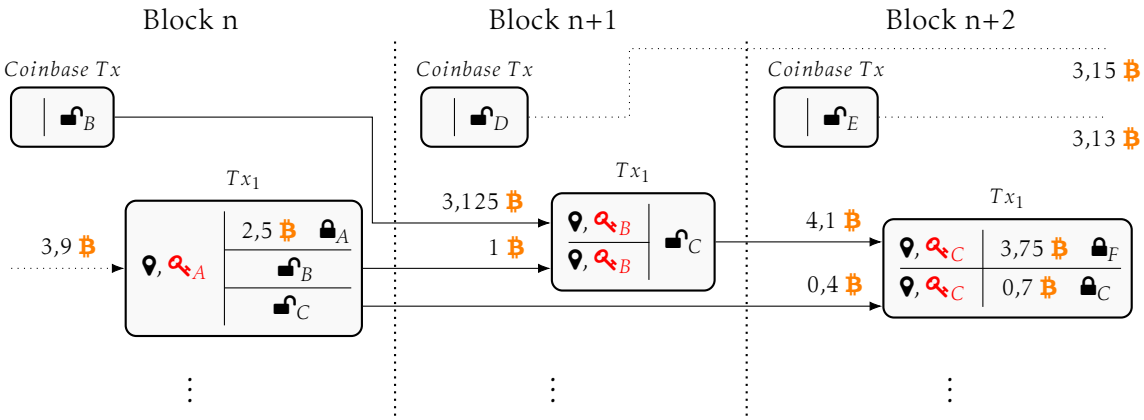


Abbildung 3.2: *Transaktionen im Überblick*

Für jeden Transaktions-Input muss zwingend eine Kante in den Knoten führen, da die Transaktion sonst ungültig ist. Hat ein Knoten keine eingehenden Kanten, also keine Inputs, handelt es sich um eine sogenannte Coinbase-Transaktion. In einer solchen Transaktion werden neue Bitcoins generiert und die gesamten Transaktionsgebühren (Fees) des Blocks gesammelt. Warum und an wen diese Beträge vergeben werden, wird im Abschnitt 5.3.5 genauer erläutert.
Ein Output einer Transaktion repräsentiert eine bestimmte Menge an Bitcoins, die dem jeweiligen Empfänger zur Verfügung stehen, um sie auszugeben. Pro Output kann maximal eine Kante aus dem Knoten führen[1]. Existiert eine solche Kante, wurde der entsprechende Betrag bereits ausgegeben. Fehlt sie, ist der entsprechende Transaktions-Output ein UTxO, der in einer späteren Transaktion ausgegeben wird.
Der Graph aller Transaktionen bildet somit einen gerichteten, azyklischen Graphen mit gewichteten Kanten. Dieser ist azyklisch, da Bitcoins nur von einer zeitlich früheren Transaktion zu einer späteren übertragen werden können. Der Graph muss nicht zusammenhängend sein. Ein Beispiel dafür sind Coinbase-Transaktionen, die noch nicht ausgegeben wurden[2].
Die in Abbildung 3.2 dargestellten Transaktionen sind in zeitlicher Reihenfolge angeordnet. Transaktionen, die weiter links eingezeichnet sind, wurden früher ausgeführt.
Im Folgenden werden die einzelnen Transaktionen beschrieben. Die tiefgestellten Buchstaben geben an, mit welchem öffentlichen bzw. privaten Schlüssel die Bitcoins verschlossen wurden bzw. entsperrt werden können:

- 🔒_A : Der Output wurde mit dem öffentlichen Schlüssel von Person A verschlossen
- 🔑_A : Person A ist im Besitz des privaten Schlüssel, der 🔒_A entsperren kann.

[1] Andernfalls würde eine doppelte Ausgabe vorliegen.
[2] Entspricht einem Knoten ohne ein- und ausgehenden Kanten

Die Coinbase-Transaktion im Block n wurde bereits von Person B ausgegeben. Dieser Output hatte einen Wert von 3,125 BTC, was dem aktuellen Blockzuschuss[3] entspricht.
Die Transaktion Tx_1 im Block n gibt 3,9 BTC aus einer nicht dargestellten Transaktion in einem vorherigen Block aus. Die Bitcoins werden auf drei Outputs verteilt, die mit den öffentlichen Schlüsseln A, B und C verschlossen wurden:

- Der erste Output mit dem Index 0 enthält 2,5 BTC und wurde noch nicht ausgegeben. Dieser Output ist ein UTxO.
- Die beiden anderen Outputs wurden bereits in späteren Transaktionen ausgegeben.

Diese Transaktion lässt sich wie folgt interpretieren: Person A möchte 1 BTC an Person B und 0,4 BTC an Person C senden. Da Person A jedoch nur einen UTxO mit einem Wert von 3,9 BTC besitzt, sendet sie 2,5 BTC als Rückgeld an sich selbst zurück.
Grundsätzlich darf die Summe der Outputs die Summe der Inputs nicht überschreiten. Es kann in einer Transaktion also nicht mehr ausgegeben werden, als eingegangen ist.

$$\sum \text{Inputs} \geq \sum \text{Outputs} \quad \Leftrightarrow \quad 3,9 \geq 2,5 + 0,4 + 1 \quad \Leftrightarrow \quad 3,9 \geq 3,9$$

Es wird ein „≥“-Zeichen verwendet, da bei Transaktionen Gebühren (Fees) anfallen können. Weitere Details zu den Transaktionsgebühren werden im Abschnitt 5.3.5.2 erläutert.
Die Coinbase-Transaktion des Blocks $n+1$ wurde bereits in einem späteren Block von Person D ausgegeben. Der Wert ist etwas höher als der aktuelle Blockzuschuss von 3,125 BTC, da in der Tx_1 in diesem Block Transaktionsgebühren bezahlt wurden.
Die Transaktion Tx_1 im Block $n+1$ hat zwei Inputs mit insgesamt 4,125 BTC. Der Output dieser Transaktion wurde bereits ausgegeben.
Die Differenz $\sum \text{Inputs} - \sum \text{Outputs} = 0,025$ BTC entspricht der bezahlten Transaktionsgebühr.
Die Coinbase-Transaktion im Block $n+2$ verläuft analog zu den vorherigen.
Die Transaktion Tx_1 dieses Blocks gibt insgesamt 4,5 BTC aus zwei vorherigen Inputs aus. Dabei sendet die Person C 3,75 BTC an Person F und 0,7 BTC als Rückgeld an sich selbst. Für diese Transaktion fielen 0,05 BTC an Gebühren an.

[3] Stand: November 2024

4 Skripte

In diesem Kapitel wird beschrieben, wie die in dem vorherigen Kapitel erläuterten UTxO technisch gesehen entsperrt und ausgegeben werden können. Die Informationen zu diesem Kapitel wurden dem Bitcoin Wiki [16] entnommen.
Bitcoin verwendet eine eigene, stack-basierte Skriptsprache[1]. Das Skript setzt sich aus dem ScriptSig und dem ScriptPubKey zusammen.
Der ScriptPubKey beschreibt die Bedingungen, unter denen die Bitcoins ausgegeben werden können. Wird das ScriptSig diesen Bedingungen gerecht, können die Bitcoins in der Transaktion ausgegeben werden. Es gibt verschiedene Arten von Skripten. In diesem Kapitel werden die beiden grundlegenden Skripte P2PK und P2PKH beschrieben. Diese gehören zu den vordefinierten Skripttypen und werden in den folgenden Unterpunkten als Standard-Skripte bezeichnet[2].

4.1 Opcodes

Opcodes sind vordefinierte Befehle, die über bestimmte Zahlen definiert sind. Beispielsweise steht `0xAC` für den Befehl `OP_CHECKSIG`.
Die verschiedenen Opcodes decken ein breites Spektrum an Funktionen ab, darunter Datenmanipulation, Kontrollfluss und kryptografische Operationen. In den folgenden Abschnitten werden die Opcodes, die in den beiden Skripten P2PK und P2PKH verwendet werden, kurz beschrieben.

4.1.1 `OP_PUSHBYTES_X`-Befehl

Die Daten eines Skriptes, wie beispielsweise eine Signatur oder ein Schlüssel, werden über `OP_PUSHBYTES_X`-Befehle[3] auf den Stack gelegt. Da diese Daten in den Skripten unterschiedliche Größen aufweisen können, muss der Bitcoin-Software mitgeteilt werden, wie viele Bytes sie auf den Stack legen soll.
Folgendes Beispiel zeigt, wie eine Signatur über den `OP_PUSHBYTES_72`-Befehl auf den Stack gelegt wird:

```
OP_PUSHBYTES_72
3045022100c219a522e65ca8500ebe05a70d5a49d840ccc15f2afa4ee9df783f06b2a32231
0220489a46c37feb33f52c586da25c70113b8eea41216440eb84771cb67a67fdb68c01
```

In den folgenden Abschnitten werden zur besseren Lesbarkeit die `PUSHBYTES_X`-Befehle und die zugehörigen Daten weggelassen und lediglich die entsprechenden Symbole (im

[1] Diese ist nicht Turing-vollständig, da beispielsweise keine Schleifen erstellt werden können.
[2] Über Updates in Bitcoin wurden neue Skripte ermöglicht, die weniger Speicherplatz benötigen und dadurch geringere Transaktionskosten aufweisen. Deshalb werden mittlerweile meist diese verwendet. Es gibt aber immer noch Transaktionen die P2PKH verwenden
[3] Das „X“ steht hier für die genaue Anzahl an Bytes, die auf den Stack gelegt werden.

obigen Beispiel ) verwendet, da sie für die Erklärung der Funktionsweise der Skripte nicht relevant sind.

4.1.2 OP_CHECKSIG-Befehl

Dieser Skriptbefehl überprüft die Gültigkeit einer Signatur. Eine gültige Signatur führt dazu, dass eine Eins auf den Stack gelegt wird, während bei einer ungültigen Signatur eine Null zurückgegeben wird. `OP_CHECKSIG` führt die im Abschnitt 2.3.3.2 beschriebene `verifizieren`-Funktion für digitale Signaturen aus.

verifizieren(,  , Nachricht) = gültig/ungültig

Die Nachricht, die an die `verifizieren`-Funktion übergeben wird, muss auf die gleiche Weise erstellt werden wie die Nachricht, die der `signieren`-Funktion übergeben wurde. In Bitcoin wird der Hashwert der Transaktionsdaten bestehend aus Inputs und Outputs ohne die enthaltenen Signaturen, signiert. Anschließend wird die erstellte Signatur in die Transaktionsdaten eingefügt. Da die Transaktion unterschrieben wird, können Ihre Daten nicht geändert werden, ohne damit die Signatur ungültig zu machen. Dadurch wird sichergestellt, dass niemand im Netzwerk Transaktionen ändern kann, bevor sie in die Blockchain aufgenommen werden.

4.1.3 Weitere Opcodes

In Tabelle 4.1 werden die verbleibenden Opcodes beschrieben, die in den beiden Skripten P2PK und P2PKH verwendet werden.

Opcode	Hex	Beschreibung
`OP_DUP`	`0x76`	Dupliziert das oberste Element des Stacks.
`OP_HASH160`	`0xa9`	Nimmt das oberste Element des Stacks, berechnet dessen HASH160-Hashwert und platziert diesen oben auf dem Stack.
`OP_EQUALVERIFY`	`0x88`	Überprüft, ob die beiden obersten Elemente des Stacks identisch sind. Sind die beiden Elemente nicht gleich, so wird das Skript vorzeitig beendet. Falls beide Elemente identisch sind, passiert nichts weiteres.

Tabelle 4.1: *Weitere Opcodes und deren Beschreibung*

4.2 Skript Validierung

Ein Skript gilt als erfolgreich ausgeführt, wenn das einzige verbleibende Element auf dem Stack einen Wert ungleich Null aufweist[4].
Ein Skript gilt als nicht erfolgreich ausgeführt, wenn eine der folgenden Bedingungen zutrifft:

- Der Stack ist am Ende der Ausführung leer.
- Das verbleibende Element auf dem Stack hat den Wert Null.
- Mehr als ein Element verbleibt am Ende der Ausführung auf dem Stack.
- Das Skript endet vorzeitig, das heißt, ohne den letzten Befehl zu erreichen.

4.3 P2PK

Das P2PK-Skript wurde in der Anfangszeit von Bitcoin verwendet und stellt das einfachste Standard-Skript dar. Die Funktionsweise wird in der nachfolgenden Tabelle 4.2 beschrieben.

Stack	Skript	Beschreibung
	 OP_CHECKSIG	ScriptSig () und ScriptPubKey (, OP_CHECKSIG) zu einem Skript kombinieren.
	**OP_CHECKSIG**	Die Signatur und der öffentliche Schlüssel werden auf den Stack gelegt.
1		Die Signatur wird für die oberen beiden Elemente des Stacks geprüft. Das Skript wurde erfolgreich ausgeführt, da das letzte verbleibende Element des Stacks eine „1“ ist.

Tabelle 4.2: *Funktionsweise P2PK-Skript*

Der Nachteil dieses Skripts ist, dass der gesamte öffentliche Schlüssel im ScriptPubKey hinterlegt werden muss. Öffentliche Schlüssel benötigen 65 Bytes[5] an Speicherplatz, was viel ist und reduziert werden kann. Außerdem bieten sie weniger Schutz im Falle eines Bruchs des ECDSA.

[4] In der Regel handelt es sich hierbei um den Wert Eins
[5] Wenn diese nicht komprimiert sind.

4.4 P2PKH

Dieser Skripttyp löst bereits einige Probleme des P2PK-Skripts. Wie der Name andeutet, wird hier der Hashwert des öffentlichen Schlüssels im ScriptPubKey hinterlegt (siehe Abschnitt 2.1.4.3). Sollte der ECDSA gebrochen werden, bietet die verwendete Hashfunktion einen zusätzlichen Schutz.
In der folgenden Tabelle 4.3 werden die einzelnen Schritte des P2PKH-Skriptes erklärt:

	Stack	Skript
1		$\sigma_{ScriptSig}$ $pk_{ScriptSig}$ } ScriptSig OP_DUP OP_HASH160 HASH160($pk_{ScriptPubKey}$) OP_EQUALVERIFY OP_CHECKSIG } ScriptPubKey
2	$pk_{ScriptSig}$ $\sigma_{ScriptSig}$	**OP_DUP** OP_HASH160 HASH160($pk_{ScriptPubKey}$) OP_EQUALVERIFY OP_CHECKSIG
3	$pk_{ScriptSig}$ $pk_{ScriptSig}$ σ	**OP_HASH160** HASH160($pk_{ScriptPubKey}$) OP_EQUALVERIFY OP_CHECKSIG
4	HASH160($pk_{ScriptSig}$) $pk_{ScriptSig}$ σ	**HASH160**($pk_{ScriptPubKey}$) OP_EQUALVERIFY OP_CHECKSIG
5	HASH160($pk_{ScriptPubKey}$) HASH160($pk_{ScriptSig}$) $pk_{ScriptSig}$ $\sigma_{ScriptSig}$	**OP_EQUALVERIFY** OP_CHECKSIG
6	$pk_{ScriptSig}$ $\sigma_{ScriptSig}$	**OP_CHECKSIG**
7	1	

Tabelle 4.3: *Funktionsweise P2PKH-Skript*

Schritt 1: Das ScriptSig und der ScriptPubKey werden zu einem Skript kombiniert. Der Stack ist zu diesem Zeitpunkt leer.

Schritt 2: Die im ScriptSig hinterlegten Daten, bestehend aus der Signatur und dem öffentlichen Schlüssel, werden auf den Stack gelegt.

Schritt 3: Der öffentliche Schlüssel wird dupliziert. Einer wird benötigt, um zu prüfen, ob das Commitment (siehe Abschnitt 2.1.4.3) im ScriptPubKey erfüllt wird, während der andere zum Verifizieren der Signatur verwendet wird.

Schritt 4: Der Hashwert des öffentlichen Schlüssels wird berechnet.

Schritt 5: Der im ScriptPubKey hinterlegte Hashwert des öffentlichen Schlüssels wird auf den Stack gelegt.

Schritt 6: Es wird geprüft, ob der berechnete Hashwert des öffentlichen Schlüssels im ScriptSig mit dem im ScriptPubKey hinterlegten Hashwert übereinstimmt. Dies ist wichtig, da der öffentliche Schlüssel des ScriptSigs anschließend zur Verifizierung der Signatur verwendet werden muss.

Schritt 7: Durch die Verifizierung der Signatur wird geprüft, ob der Absender den privaten Schlüssel besitzt, der zu dem angegebenen öffentlichen Schlüssel gehört. In diesem Fall ist die Signatur gültig, da eine „1“ auf den Stack gelegt wurde. Das Skript ist insgesamt erfolgreich durchgelaufen, da alle Punkte des Skripts ausgeführt wurden und auf dem Stack lediglich eine „1“ liegt.

4.5 Weitere Standard-Skripte

In diesem Abschnitt werden zwei weitere wichtige Standard-Skripte im Bitcoin-Netzwerk kurz vorgestellt. Es besteht auch die Möglichkeit, eigene Skripte zu definieren. Allerdings leiten die Knoten im Bitcoin-Netzwerk nur Transaktionen mit Standard-Skripten weiter, sodass benutzerdefinierte Skripte selten in Blöcken zu finden sind.

Pay To Script Hash (P2SH):

Im ScriptPubKey[6] ist kein öffentlicher Schlüssel oder dessen Hashwert hinterlegt, sondern der Hashwert eines Skriptes. Das ScriptSig enthält eine oder mehrere Signaturen sowie das vollständige Skript, dessen Hashwert mit dem im ScriptPubKey übereinstimmen muss. Dies ermöglicht die Erstellung komplexer Skripte, während weiterhin Bitcoins an einfache Adressen (siehe Abschnitt 4.6) gesendet werden können. P2SH ermöglicht beispielsweise Transaktionen, die von mehreren Parteien autorisiert werden müssen. [17, S. 151–153]

Pay To Witness Public Key Hash (P2WPKH):

[6] ScriptPubKey = `HASH160 <Skrtipt-Hash> OP_EQUALVERIFY`

Dieses Skript funktioniert ähnlich wie das P2PKH-Skript. Jedoch werden die Skriptdaten nicht mehr im ScriptSig abgelegt, sondern in einem neuen Feld außerhalb der eigentlichen Transaktionsdaten, wodurch sie die TxID nicht beeinflussen. Durch diesen Ansatz konnte das Problem der „Transaction Malleability“ gelöst werden. [17, S. 330]

4.6 Adressen

Eine Bitcoin-Adresse ist eine benutzerfreundliche Darstellung eines ScriptPubKeys und wird nicht direkt in Bitcoin-Transaktionen verwendet, weshalb sie auch nicht in der Blockchain erscheint. Stattdessen dienen Adressen dazu, das Senden von Bitcoins über Wallets zu erleichtern. Eine Adresse ist ein ScriptPubKey, der entweder mit `Base58` oder `Bech32` kodiert wurde. Es gibt verschiedene Typen von Adressen, die zu verschiedenen ScriptPubKeys dekodiert werden können. Abbildung 4.1 zeigt für einen P2PKH ScriptPubKey, wie er in eine Adresse umgewandelt wird und umgekehrt.
Der Skript-Typ bestimmt die Versionsnummer. Bei P2PKH ist das beispielsweise $0x00$. Die Prüfsumme entspricht den ersten 4 Bytes des Hashwerts von $HASH256$(Version + 🔑). Die P2PKH Adresse wird berechnet über:
$Base58$(Versionsnummer + HASH160(🔑) + Prüfsumme).
Adressen werden häufig auf Webseiten geteilt, um es anderen zu erleichtern, Bitcoins zu senden.

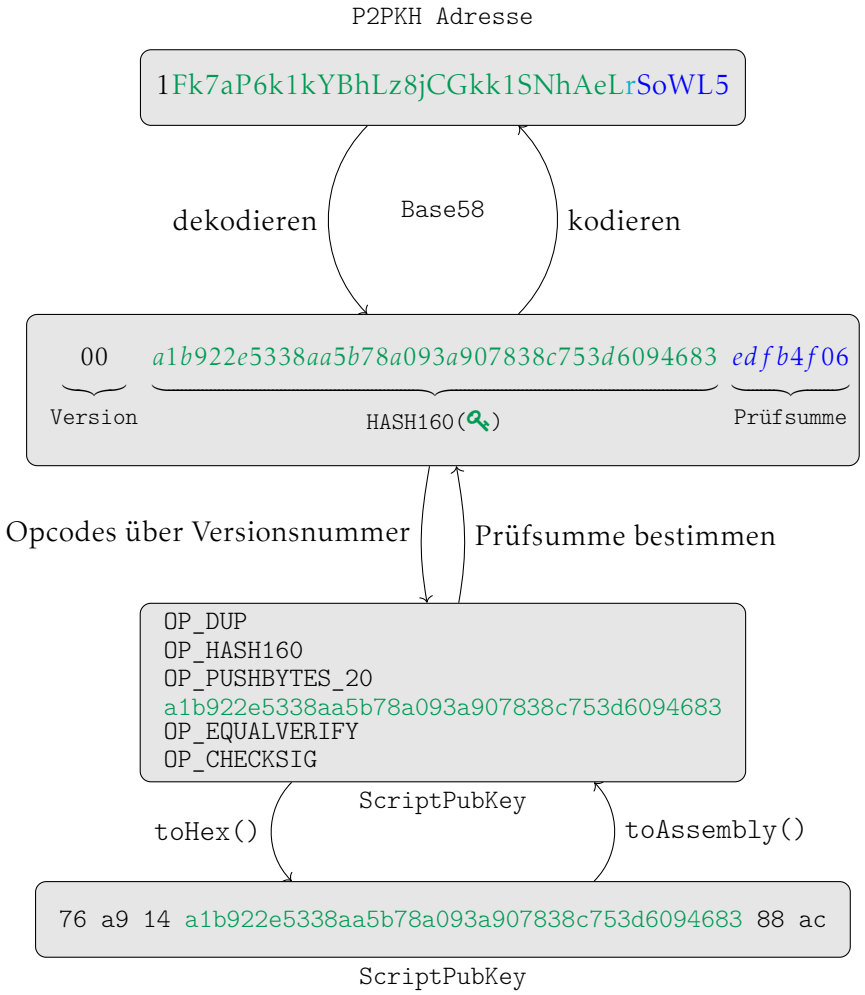


Abbildung 4.1: *Umwandlung zwischen P2PKH-Adresse und ScriptPubKey*

5 Blockchain

Die in diesem Kapitel erklärten Konzepte basieren auf dem Whitepaper „Bitcoin: A Peer-to-Peer Electronic Cash System“ [15]. Die technischen Details entstammen der Webseite learnmeabitcoin.com [18]. Sofern nicht anders angegeben, wird auf diese beiden Quellen Bezug genommen.
Es wird die von Satoshi Nakamoto vorgeschlagene Lösung für das sogenannte „Double-Spending“-Problem in einem genehmigungslosen[1] peer-to-peer (P2P)-Netzwerk vorgestellt. Wie bereits im Kapitel Transaktionen beschrieben, kann das doppelte Ausgeben von Bitcoins verhindert werden, wenn die Transaktionen in zeitlicher Reihenfolge vorliegen. Auf diese Weise kann in allen Transaktionen überprüft werden, ob die auszugebenden Bitcoins bereits in einer anderen Transaktion ausgegeben wurden.
Bitcoin fand als erstes eine praktisch umsetzbare Lösung, wie sich die Teilnehmer eines genehmigungslosen P2P-Netzwerk auf eine zeitlich geordnete Transaktionshistorie einigen, die vor Veränderungen gesichert ist bzw. Veränderungen im Nachhinein erkennt. Diese neu mit Bitcoin eingeführte Lösung ist die Blockchain, die die folgenden Eigenschaften aufweist [19]:

Dezentral: Alle Teilnehmer des Netzwerks speichern eine eigene Kopie der Blockchain und aktualisieren diese stetig.

Transparenz: Die Daten der Blockchain können von allen gelesen werden. Um jedoch die Privatsphäre der Nutzer zu schützen, werden Schlüsselpaare nur einmal verwendet, wobei für jede Transaktion ein neues Schlüsselpaar generiert wird. So bleibt die Identität der Nutzer anonym, während die Nachvollziehbarkeit der Transaktionen gewährleistet ist.

Verkettung und manipulationssicher: Die einzelnen Blöcke der Blockchain sind über Hash-Pointer miteinander verkettet, wodurch sie manipulationssicher werden. Dies wird in Abschnitt Abschnitt 5.2 genauer beschrieben.

Konsensmechanismus: Ein Mechanismus, der es ermöglicht, dass sich alle Teilnehmer auf eine gültige Blockchain einigen. Der Konsnensmechanismus wird in Abschnitt 5.3.2 beschrieben und wie dadurch ein verteilter Konsens erriecht wird ist in Abschnitt 5.4 beschrieben.

In den folgenden Abschnitten wird zunächst der grundsätzliche Aufbau der Blockchain erklärt, indem detailliert dargestellt wird, wie ein einzelner Block aussieht und weshalb die Blöcke miteinander verkettet werden. Anschließend wird erläutert, wie neue Blöcke zur Blockchain hinzugefügt werden und wie dadurch gewährleistet wird, dass sich alle Teilnehmer des Netzwerks auf die gleiche Blockchain einigen.

[1] Jeder kann ohne Einschränkungen am Netzwerk teilnehmen.

5.1 Block

Ein Block speichert eine begrenzte Anzahl von Transaktionen. Der Blockheader fungiert dabei als eine Art Zusammenfassung des gesamten Blocks. Der Blockaufbau kann der Abbildung 5.1 entnommen werden.

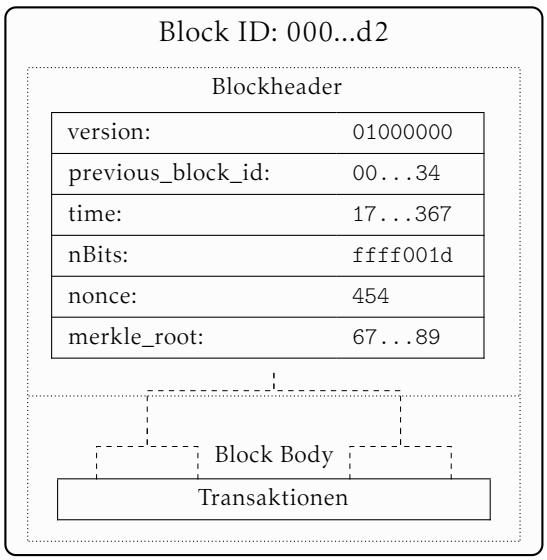


Abbildung 5.1: *Aufbau eines Blocks in Bitcoin*

`version`: Die Versionsnummer des Blocks, die zur Kennzeichnung der aktuellen Software-Version dient.

`previous_block_id`: Dieses Feld enthält den Hash-Wert des vorherigen Blocks. Dadurch werden die Blöcke über Hash-Pointer miteinander verkettet, was sicherstellt, dass der referenzierte Block zeitlich vor dem aktuellen liegt.

`time`: Der Zeitstempel des aktuellen Blocks. Dieser muss innerhalb eines Toleranzbereichs von ± 2 Stunden liegen, damit der Block vom Netzwerk akzeptiert wird. Der Zeitstempel dient hauptsächlich zur Anpassung des `targets` (siehe Abschnitt 5.3.4).

`nBits`: Über diesen Wert kann das `target` berechnet werden (siehe Abschnitt 5.3.1).

`nonce`: Number used only once (Nonce), die während des Mining-Prozesses verwendet wird, um die Block ID zu ändern, ohne die eigentlichen Blockdaten ändern zu müssen (siehe Abschnitt 5.3.2).

`merkle_root`: Ein Hashwert der alle Transaktionen im Block repräsentiert (siehe Abschnitt 2.2). Dieses Feld verbindet den Blockheader mit den gespeicherten Daten im Blockbody.

Block Body: Hier sind die Transaktionen des Blocks gespeichert.

Die Block ID identifiziert einen Block innerhalb der Blockchain eindeutig. Sie wird durch die Berechnung des HASH256-Hashwerts des Blockheaders erzeugt. Über sie wird die Datenintegrität der Blöcke sichergestellt. Für die Gültigkeit eines Blocks muss die Block ID kleiner sein als das `target`.

5.2 Verkettung und Manipulationssicherheit

Ein Block enthält stets den Hashwert des vorherigen Blocks. Dadurch bilden die Blöcke eine Kette wie in Abbildung 5.2 zu sehen ist und erhalten eine feste zeitliche Reihenfolge. Die Idee zur Verkettung über Hashwerte entnahm Satoshi Nakamoto dem Artikel „How To Time-Stamp a Digital Document“ von Haber und Stornetta [20]. Aus dieser Verkettung von Blöcken entstand der Begriff 'Blockchain'. Im ursprünglichen Bitcoin-Whitepaper wird dieser Begriff jedoch nicht explizit verwendet, er hat sich erst später etabliert.

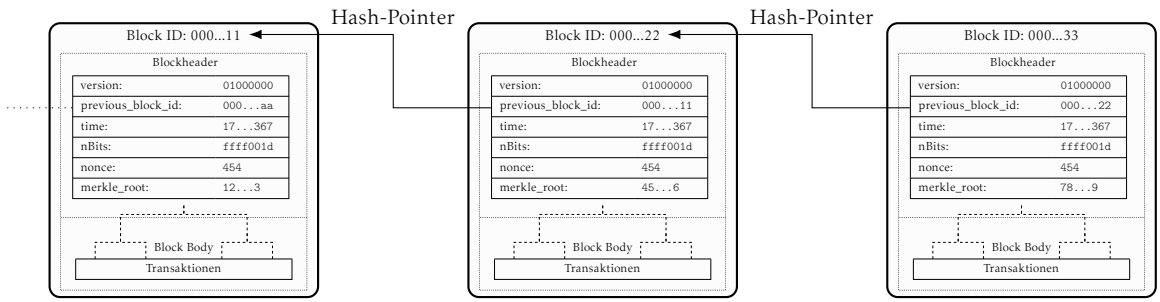


Abbildung 5.2: *Blöcke über Hash-Pointer verkettet*

In Abbildung 5.3 wurde eine Transaktion verändert. Damit das Feld `merkle_root` die veränderten Daten widerspiegelt, muss der Hash-Baum neu berechnet werden. Der neue Wert für die `merkle_root` führt zu einer Änderung der Block ID. Damit der Nachfolgende Block auf diesen veränderten Block verweist, muss das Feld `pervious_block_id` des nachfolgenden Blocks aktualisiert werden, was wiederum dessen Block ID verändert.
Durch diese eine Änderung wird der betroffene Block und alle nachfolgenden Blöcke ungültig, da ihre Block IDs nicht mehr kleiner sein werden als der erforderliche `target`-Wert (siehe Abschnitt 2.1.3.1). Unabhängig davon, welche Daten innerhalb eines Blocks verändert werden, wirkt sich die Änderung auf die Block ID dieses Blocks sowie auf alle nachfolgenden Blöcke aus.

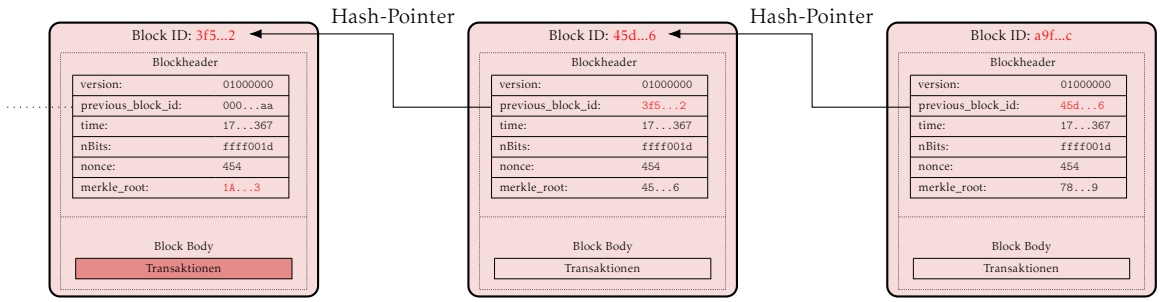


Abbildung 5.3: *Datenintegrität der Blockchain verdeutlicht*

5.3 Mining

In diesem Abschnitt wird erläutert, wie Miner (im Deutschen auch „Schürfer“ genannt) einen neuen Block an die Blockchain anhängen können und welchen Anreiz sie dafür haben.
Jeder Miner verwaltet seinen eigenen Memory Pool (Mempool), in dem er Transaktionen speichert, die er zuvor validiert hat und die noch nicht in einen Block aufgenommen wurden. Dieser Pool dient als temporärer Speicher für Transaktionen, bis sie in die Blockchain geschrieben oder nach einer gewissen Zeit verworfen werden. Miner wählen Transaktionen aus dem Mempool aus, um einen sogenannten „Candidate Block“ zu erstellen, den sie dann versuchen, an die Blockchain anzuhängen.

5.3.1 Target

Das `target` beschränkt den möglichen Bereich der Block IDs, indem es eine Obergrenze auf dem Zahlenstrahl zwischen 0 und 2^{256} definiert. Eine gültige Block ID muss kleiner als diese Obergrenze sein. Wegen der Gleichverteilung von Hashwerten, entspricht die Wahrscheinlichkeit, dass eine Block-ID unterhalb des `targets` liegt, dem Verhältnis der möglichen Werte unterhalb des `target` zu allen möglichen Hashwerten. Dieses Verhältnis bestimmt letztlich, wie schwierig es ist, einen gültigen Block zu finden.

$$\mathrm{P}(\text{Block ID} < \texttt{target}) = \frac{|\{0, \ldots, \text{target - 1}\}|}{|\{0, \ldots, 2^{256} - 1\}|} = \frac{\texttt{target}}{2^{256}}$$

Technisch gesehen ist das `target` eine 64 Byte große Hexadezimalzahl, das über das 4 Byte große `nBits`-Feld des Blockheaders bestimmt werden kann. Die Umrechnung geht wie folgt:

- Das erste Byte des `nBits`-Feldes entspricht dem Exponenten.
- Die verbleibenden drei Bytes bilden die Mantisse.

Das `target` wird dann berechnet durch:

$$\texttt{target} = \text{Mantisse} \cdot 256^{(\text{Exponent - 3})}$$

Diese Umrechnung kann auch als Bitverschiebung interpretiert werden: Der Exponent gibt an, um wie viele Bytes die Mantisse nach links verschoben wird[2].

nBits: 0x170f48e4

Exponent: $0x17 = 23_2$

Mantisse: 0x0f48e4

target: $0x0000000000000000\underbrace{\mathbf{0f48e4}0000000000000000000000000000000000000000}_{\text{Mantisse 23 Bytes nach links verschieben}}$

5.3.2 Mining-Prozess

Die Miner berechnen die Block ID für ihren Candidate Block und prüfen, ob diese kleiner ist als das `target`. Ist dies nicht der Fall, inkrementieren sie die Nonce und berechnen die Block ID erneut. Sobald ein Miner eine Block ID findet, die kleiner als das `target` ist, gilt der Block als gültig und kann an die Blockchain angehängt werden. Je mehr Rechenleistung einem Miner zur Verfügung steht, desto häufiger kann er eine neue Block ID berechnen und desto wahrscheinlicher wird es, dass er einen gültigen Block findet. Das `target` ist dabei so gewählt, dass im Durchschnitt alle 10 min ein Miner im Netzwerk einen Block erstellt.
Während das Finden eines gültigen Blocks sehr rechenintensiv ist, lässt sich die Gültigkeit eines Blocks effizient in $\mathcal{O}(1)$ prüfen. Anders ausgedrückt, es lässt sich leicht feststellen, dass die Miner die erforderliche Arbeit erbracht haben. Der Mining-Prozess wird auch als „Proof of Work“ bezeichnet.
Ändert sich die Block-ID eines Blocks durch Veränderung dessen Daten, wie in Abbildung 5.3, müsste die gesamte Arbeit erneut verrichtet werden, um eine neue gültige Block ID zu finden. Dasselbe gilt für alle nachfolgenden Blöcke.

5.3.3 Schwierigkeit (Difficulty)

Das `target`, das für den ersten Block (Genesis Block) verwendet wurde, wird als maximales `target` definiert.
Die Schwierigkeit gibt an, wie schwer es ist, einen neuen Block zur Blockchain hinzuzufügen, im Vergleich zum ersten Block.

$$\texttt{target} = \frac{\text{maximales } \texttt{target}}{\text{Schwierigkeit}} \Leftrightarrow \text{Schwierigkeit} = \frac{\text{maximales } \texttt{target}}{\texttt{target}}$$

Die Schwierigkeit und das `target` sind indirekt proportional zueinander. Steigt die Schwierigkeit, so fällt das `target`. Dies ergibt Sinn, da das `target` die möglichen Werte der Block-ID beschränkt. Da jeder Hashwert gleich wahrscheinlich ist, wird es mit einem kleiner werdenden `target` schwieriger, eine gültige Block-ID zu finden.
Letztendlich ist die Schwierigkeit aber nur eine alternative Darstellung des `targets`.

[2] 1 Byte entspricht zwei hexadezimal Stellen, deshalb wird die Mantisse in diesem Beispiel um 46 Stellen verschoben.

5.3.4 Target-Anpassung

Die Schwierigkeit wird durch das Bitcoin-Protokoll alle 2016 Blöcke[3] automatisch angepasst. In diesem Abschnitt wird gezeigt, wie sich das `target` automatisch an die Rechenleistung im Netzwerk anpasst.
Das `target` wird so eingestellt, dass die durchschnittlich Blockzeit[4] wieder 10 Minuten beträgt. Diese Anpassung ist notwendig, wenn sich die Rechenleistung im Bitcoin-Netzwerk verändert hat. Zur Berechnung des neuen `targets` wird der alte Wert mit dem Quotienten aus der tatsächlich benötigten und der erwarteten Zeit multipliziert. Die tatsächliche Zeit wird über den Zeitstempel `time` im Blockheader bestimmt.

$$\texttt{target}_{neu} = \frac{\text{tatsächlich benötigte Zeit}}{\text{erwartete Zeit}} \cdot \text{target}_{alt}$$

Beispielhafte Anpassung des `targets`:

Gegeben:

- Alte Wert des `targets`: 0xf48e400000000000000000000000000000000000000000
- Die letzten 2016 Blöcke haben durchschnittlich 8 min zum Erstellen benötigt.

Berechnung des neuen `targets`:

$$\frac{2016 * 8\ \text{min}}{2016 * 10\ \text{min}} \cdot \texttt{target}_{alt} = \text{0xc3a500000000000000000000000000000000000000000000}$$

5.3.5 Blockbelohnung (Block Reward)

> „By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them“ [15]

Die spezielle erste Transaktion eines Blocks wird Coinbase-Transaktion genannt (in Abbildung 3.2 zu sehen). In dieser Transaktion erhält der Miner eines Blocks eine Belohnung für seine erfolgreiche Arbeit, die aus dem Blockzuschuss (Block Subsidy) und den Transaktionsgebühren (Fees) besteht. Diese Belohnung kompensiert die aufgewendete CPU-Zeit und den Stromverbrauch der Miner und schafft einen Anreiz, neue Blöcke zu erstellen und das System sicher und funktionsfähig zu halten. Da in dieser Transaktion neue Bitcoins erschaffen werden, besitzt sie einen leeren Input. Abgesehen davon ist sie wie jede andere Transaktion aufgebaut.

[3] Es dauert ungefähr 2 Wochen um 2016 Blöcke zu erstellen
[4] Die Zeit die benötigt wird, um einen neuen Block zu minen

5.3.5.1 Blockzuschuss (Block Subsidy)

Über den Blockzuschuss gelangen neue Bitcoins in das Netzwerk. Die Menge an Bitcoins, die für das Erstellen eines Blocks neu erschaffen werden, hängt von der aktuellen Blockhöhe ab. Zu Beginn erhielten die Miner 50 BTC pro Block. Alle 210 000 Blöcke wird dieser Zuschuss halbiert, ein Ereignis, das in der Bitcoin-Community als „Halving" bekannt ist. Aufgrund dieser Halbierung ist die Gesamtanzahl der Bitcoins begrenzt und kann theoretisch wie folgt berechnet werden:

$$(50\ \text{BTC} \cdot 210\,000 + 25\ \text{BTC} \cdot 210\,000 + 12.5\ \text{BTC} \cdot 210\,000 + \ldots) =$$
$$50\ \text{BTC} \cdot 210\,000 \cdot (1 + \frac{1}{2} + \frac{1}{4} + \ldots) = 50\ \text{BTC} \cdot 210\,000 \cdot \sum_{n=0}^{\infty} \frac{1}{2^n} = 21\,000\,000\ \text{BTC}$$

Dieser Wert kann in der Praxis nicht erreicht werden, da ein Bitcoin nicht unendlich oft halbiert werden kann. In Bitcoin entspricht die kleinste Einheit, benannt nach dem Erfinder, einem Satoshi: 1×10^{-8} BTC = 1 SAT. Es ergibt sich eine in der Praxis maximale Anzahl von Bitcoins, die wie folgt berechnet wird:

$$210\,000 \cdot \sum_{n=0}^{\infty} \left(\left\lfloor 5\,000\,000\,000\ \text{SAT} \cdot \frac{1}{2^n} \right\rfloor \right) =$$
$$2\,099\,999\,997\,690\,000\ \text{SAT} = 20\,999\,999.9769\ \text{BTC}$$

In Abbildung 5.4 sind der Blockzuschuss sowie die aktuelle Anzahl der im Umlauf befindlichen Bitcoins in Abhängigkeit von der Blockhöhe dargestellt.

5.3.5.2 Transaktionsgebühren (Fees)

Erinnern wir uns zurück an folgende grundlegende Bedingung für eine valide Transaktion:

$$\sum \text{Inputs} \geq \sum \text{Outputs}$$

Die Differenz zwischen den Inputs und Outputs einer Transaktion entspricht den bezahlten Transaktionsgebühren und steht dem Miner über die Coinbase-Transaktion zur Verfügung:

$$\sum \text{Inputs} - \sum \text{Outputs} = \text{Fee}$$

Miner stellen ihren Kandidatenblock in der Regel so zusammen, dass die Transaktionsgebühren maximiert werden. Je höher die von einem Nutzer festgelegten Transaktionsgebühren, desto wahrscheinlicher ist es, dass die Transaktion in einem Block aufgenommen wird. Wählt man hingegen eine zu geringe Transaktionsgebühr, besteht die Gefahr, dass die Transaktion nicht in einen Block aufgenommen wird und nach einer bestimmten Zeit von den Knoten im Netzwerk verworfen wird.
Transaktionsgebühren spielen zudem eine wichtige Rolle, wenn der Blockzuschuss in Zukunft zu gering wird oder vollständig entfällt, da sie den Anreiz für Miner schaffen, weiterhin neue Blöcke zu erzeugen.

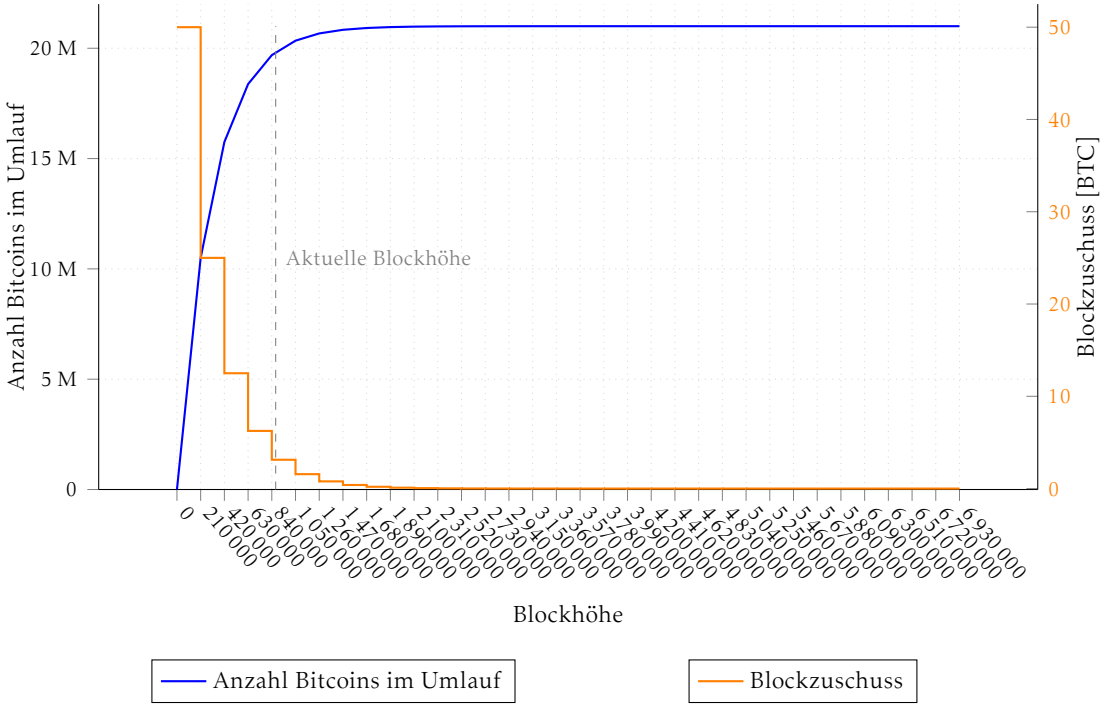


Abbildung 5.4: *Blockzuschuss und Anzahl Bitcoins in Abhängigkeit von der Blockhöhe*

5.4 Konsens

> „The proof-of-work also solves the problem of determining representation in majority decision making. [...] Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains“ [15]

Die Teilnehmer des Netzwerkes einigen sich über eine gültige Transaktionshistorie, indem sie immer der Blockchain mit der meisten kumulativen Rechenleistung vertrauen. Es ist als würden die Miner im Netzwerk mit Ihrer Rechenleistung abstimmen, welcher Blockchain vertraut werden soll. Und solange über die Hälfte der Rechenleistung von ehrlichen Miner besitzt wird, wird deren Kette immer länger werden als die der unehrlichen.

5.5 Ketten-Reoganisation

Wenn zwei Blöcke gleichzeitig geschürft werden, kann es passieren, dass Teilnehmer des Netzwerks unterschiedliche Blöcke als den nächsten in der Kette haben. Zunächst vertraut jeder dem Block, den er zuerst erhalten hat, und speichert den anderen vorübergehend ab. Würde für den anderen Block zuerst ein nachfolgender Block erzeugt werden, so würde man zu dieser Kette wechseln, da in dieser nun mehr Rechenleistung steckt. Dieser Prozess wird als Ketten-Reorganisation bezeichnet.

6 Das Bitcoin Netzwerk

Dieses Kapitel basiert auf den Informationen aus dem Buch „Mastering Bitcoin“ von Andreas M. Antonopoulos [17].
Das Bitcoin-Netzwerk besteht aus Knoten, die die Bitcoin-Software ausführen. Die ausgeführte Software ist der Bitcoin Client. Es gibt die offizielle bitcoin-core Software, aber auch alternative Software die benutzt werden kann, es kann sogar selber geschriebene Software verwendet werden. Jeder Knoten im Netzwerk benötigt lediglich eine Internetverbindung, um mit anderen Knoten Daten auszutauschen und so Teil des Bitcoin-Netzwerks zu werden.
Das Bitcoin-Netzwerk ist ein Peer-to-Peer- (P2P) Netzwerk [15], was bedeutet, dass alle Knoten miteinander verbunden (Netzwerk) sind und gleichberechtigt (peers) im Netzwerk agieren.

6.1 Knoten im Bitcoin-Netzwerk

Ein vollwertiger Knoten im Bitcoin-Netzwerk hat drei wesentliche Aufgaben.

1. Daten im Netzwerk austauschen

Innerhalb des Bitcoin-Netzwerks tauschen die Knoten Daten aus. Dabei handelt es sich beispielsweise um Nachrichten, um den Netzwerkverkehr zu ermöglichen, sowie um neu erstellte Blöcke oder auszuführende Transaktionen.

2. Daten validieren anhand der im Bitcoin-Protokoll definierten Regeln

Die folgende Auflistung zeigt einige Regeln, die bei der Validierung von Transaktionen beachtet werden müssen [21]:

- Transaktionen und Blöcke müssen die in der Bitcoin-Software definierten Formate einhalten.
- Es darf nicht mehr ausgegeben werden, als man besitzt ($\sum \text{Inputs} \geq \sum \text{Outputs}$).
- Die Transaktion muss Inputs und Outputs haben.
- Die Inputs müssen ausschließlich UTxO referenzieren.
- Die Gültigkeit der Signaturen muss geprüft werden.

Folgende Liste zeigt einige Punkte, die bei der Blockvalidierung beachtet werden. [21]:

- Prüfen, ob das `nBits`-Feld der aktuellen Schwierigkeit entspricht.
- Prüfen, ob die Block ID kleiner als das `target` ist.
- Die erste Transaktion im Block muss die Coinbase-Transaktion sein.
- Prüfen der Merkle-Root.
- Prüfen, ob der Zeitstempel aktuell genug ist.

3. Kopie der Blockchain aufbewahren

Es muss die Blockchain gespeichert werden, um zum einen die Validierung von Blöcken und Transaktionen durchführen zu können. Das speichern der Transaktionshistorie in Zusammenspiel mit dem Datenaustausch ermöglicht sämtlichen Clients auf dem aktuellen Stand zu bleiben. Es kann so auch die Blockchain an neue Teilnehmer des Netzwerks geteilt werden.

6.2 Teilnehmer im Bitcoin Netzwerk

6.2.1 Full Nodes

Eine Full Node ist ein vollwertiger Knoten im Bitcoin-Netzwerk, der die gesamte Blockchain speichert und alle eingehenden Daten validiert. Da eine Full Node alle Daten selbst speichert, muss sie nicht auf Dritte vertrauen, wie es bei Lightweight Nodes (siehe Abschnitt 6.2.2) der Fall ist. Eine Full Node benötigt mehr Speicherplatz, kann sich dafür sehr sicher sein, dass diese korrekt sind.
Full Nodes können Speicherplatz zurückgewinnen, indem sie ausgegebene Transaktionen löschen. Transaktionen, die keine UTxOs mehr enthalten, sind nicht mehr notwendig und können entfernt werden. In diesem Zusammenhang könnte man sich vorstellen, dass die in Abbildung 2.6 nicht eingezeichneten Transaktionen keine UTxO enthalten und daher gelöscht wurden. In solchen Fällen müssten nur die verbleibenden Transaktionen und deren Merkle-Proof gespeichert werden.
Das Löschen von Transaktionen ist jedoch nur dann sinnvoll, wenn diese sich in Blocks weiter hinten in der Blockchain befinden. In diesen Fällen kann man sicher sein, dass sich die Transaktionen nicht mehr ändern, da die Wahrscheinlichkeit einer Änderung eines Blocks mit zunehmender Tiefe in der Blockchain exponentiell fällt (siehe Kapitel 7). [15]

6.2.2 Simplified Payment Verification

Dieser Abschnitt beschreibt den Simplified Payment Verification (SPV) Knoten, der er im Whitepaper „Bitcoin: A Peer-to-Peer Electronic Cash System“ von Nakamoto [15] vorgestellt wird.
Ein solcher Knoten speichert lediglich die Blockheader und nicht die gesamte Transaktionshistorie, um Speicherplatz zu sparen. Der Knoten kann die Längste Kette von Full Nodes abfragen, er kann jedoch jedoch keine Blöcke und Transaktionen die er empfängt selbst validieren. Ein SPV-Knoten kann über die Merkle-Root den Merkle-Proof (siehe Abbildung 6.1 und Abbildung 2.6) von einer Full Node anfragen und prüfen, ob eine Transaktion in einem Block vorhanden ist. Solange es mehr ehrliche Full Nodes als unehrliche gibt, kann ein SPV-Knoten davon ausgehen, dass die erhaltenen Daten der Full Nodes korrekt sind.

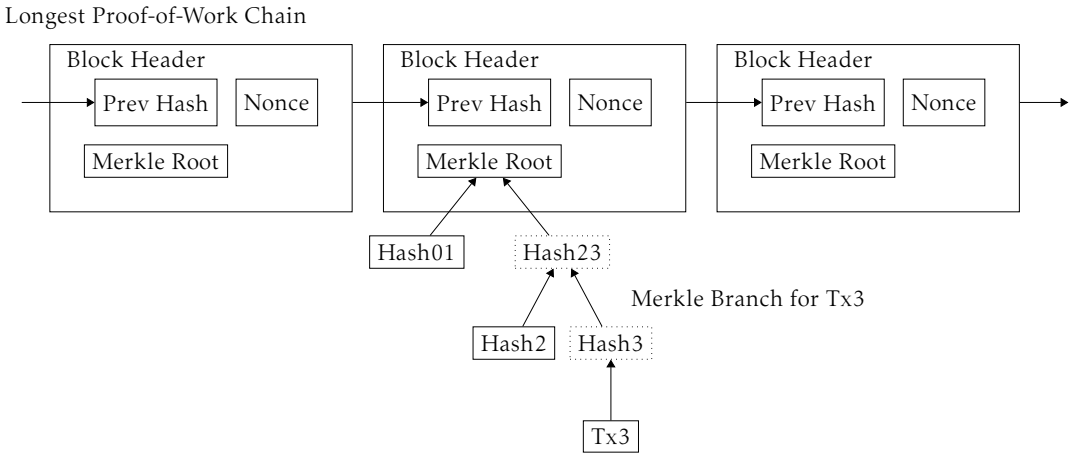


Abbildung 6.1: *Simplified Payment Verification. Quelle: [15]*

6.2.3 Miner

Ein Miner ist in der Regel eine Full Node, muss es jedoch nicht zwingend sein. Ein Miner versucht, neue Blöcke für die Blockchain zu erstellen und sendet diese an die Knoten des Netzwerks.

7 51 %-Attacke

Es gibt Angriffsszenarien, die das Bitcoin-Netzwerk gefährden können. In diesem Abschnitt wird das in [15, S. 6–8] beschriebene Szenario erläutert, indem ein Angreifer eine alternative Blockchain mit mehr Rechenleistung erstellt, um die bestehende Transaktionshistorie umzuschreiben. Dieser Angriff wird als 51 %-Attacke bezeichnet. Auch wenn es technisch möglich ist, die Transaktionshistorie nachträglich zu ändern, kann ein Angreifer weder frei neue Bitcoins erschaffen noch sich fremde Bitcoins aneignen. Dies ist deshalb nicht möglich, weil er an die Konsensregeln des Bitcoin-Protokolls gebunden ist, die in der Bitcoin-Software implementiert sind. Verstöße gegen diese Regeln würden dazu führen, dass die manipulierte Blockchain von den anderen Knoten im Netzwerk abgelehnt wird. Möglich ist jedoch, dass ein Angreifer eine zuvor getätigte Transaktion rückgängig macht, um bereits ausgegebene Bitcoins zurückzuerlangen.
Zunächst werden folgende Wahrscheinlichkeiten definiert:

p = Wahrscheinlichkeit, dass ein ehrlicher Schürfer den nächsten Block schürft

q = Wahrscheinlichkeit, dass der Angreifer den nächsten Block erzeugt

q_z = Wahrscheinlichkeit, dass der Angreifer einen z Block Rückstand aufholt

$$q_z = \begin{cases} 1 & \text{wenn } p \leq q \\ \left(\frac{q}{p}\right)^z & \text{wenn } p > q \end{cases}$$

Beispielrechnung:
Angenommen, der Angreifer verfügt über 30 % der gesamten Rechenleistung des Netzwerks und hat einen Rückstand von $z = 10$ Blöcken. Die Wahrscheinlichkeit, dass er diesen Rückstand aufholt, berechnet sich wie folgt:

$$q_z = \left(\frac{0,3}{1-0,3}\right)^{10} = 0.000\,209 \approx 0,02\%$$

Die Wahrscheinlichkeit sinkt exponentiell mit der Anzahl an Blöcken die aufgeholt werden müssen. Daher sind ältere Blöcke praktisch unveränderbar.
Besitzt der Angreifer mehr als 50 % der gesamten Rechenleistung, so steigt die Wahrscheinlichkeit eines erfolgreichen Angriffs auf theoretisch 100 %. Das bedeutet, der Angreifer könnte beliebig viele Blöcke aufholen und damit die Transaktionshistorie vollständig umschreiben. Dieser Angriff ist sehr zeitaufwendig[1], insbesondere wenn es darum geht, weit zurückliegende Blöcke zu überschreiben.
Nun wird folgendes Angriffsszenario betrachtet:

[1] Es müsste die komplette Arbeit die seit 2009 verrichtet wurde erneut aufgebracht werden, was wieder mehrere Jahre benötigen würde.

- Der Angreifer sendet Person A eine Transaktion über 1 BTC und erhält im Gegenzug den Betrag X in Euro, nachdem die Transaktion in die Blockchain aufgenommen wurde.
- Parallel dazu arbeitet der Angreifer im Hintergrund an einer alternativen Blockchain. In dieser ersetzt er die ursprüngliche Transaktion „Angreifer sendet 1 BTC an Person A“ durch eine neue Transaktion, in der er den selben Bitcoin an sich selbst überweist.
- Sobald in der alternative Blockchain des Angreifers mehr Arbeit steckt, übermittelt er diese an das Netzwerk.
- Die Knoten des Netzwerks akzeptieren diese Kette, da sie die Konsensregeln einhält und nehmen eine Ketten-Reorganisation vor (siehe Abschnitt 5.5). Die ursprüngliche Transaktion ist nicht mehr in der Blockchain enthalten und kann nicht nachträglich hinzugefügt werden, da der referenzierte Output bereits ausgegeben wurde. Person A hätte dadurch den Betrag X an den Angreifer verloren.

Nun stellt sich die Frage, wie lange ein Empfänger warten muss um ausreichend sicher zu sein, dass die Transaktion nicht mehr im Nachhinein ausgetauscht werden kann.
Anhand des konkreten Beispiels in Abbildung 7.1 wird eine allgemeine Formel ausgearbeitet, wie sicher sich ein Empfänger sein kann, dass sich ein Block nicht mehr ändert, in Abhängigkeit von der prozentualen Rechenleistung des Angreifers im Netzwerk und der Anzahl an Blöcken, die seit diesem Block neu geschürft wurden.

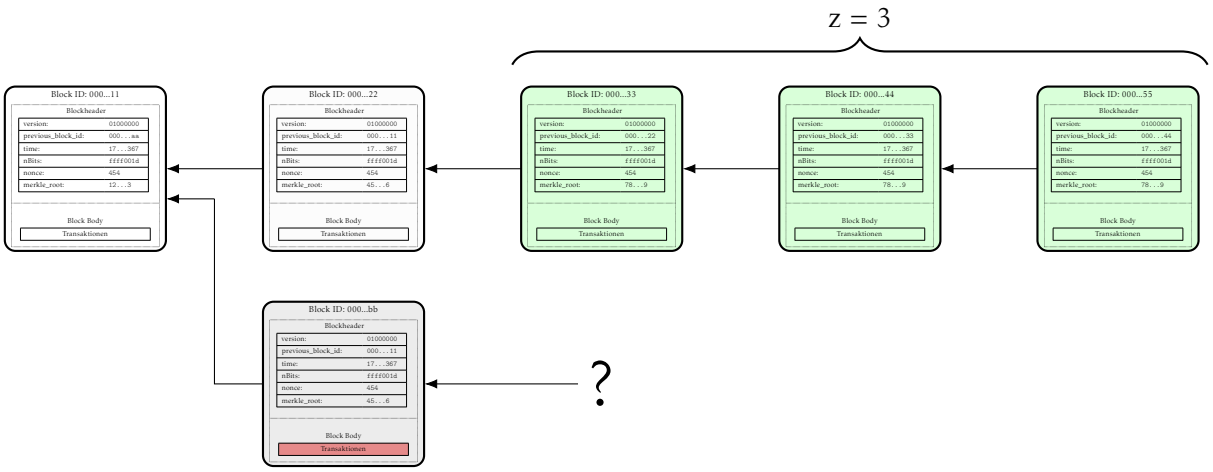


Abbildung 7.1: *51 %-Angriff*

In dem grauen Block in Abbildung 7.1 fand die beschriebene Transaktion in einen Block. Nun wurden bereits drei neue Blöcke ($z = 3$) von „ehrlichen Schürfern“ im Netzwerk erstellt. Der Empfänger weiß jedoch nicht, wie viele Blöcke der Angreifer in der Zwischenzeit erstellt hat. Die Wahrscheinlichkeit, dass der Angreifer noch aufholen kann, berechnet sich für dieses Beispiel wie folgt:

$$
\begin{aligned}
&\text{P(Angreifer hat 0 Blöcke erstellt)} \cdot \text{P(Angreifer holt 3 Blöcke auf)} \\
&+ \ldots \\
&+ \text{P(Angreifer hat 3 Blöcke erstellt)} \cdot \text{P(Angreifer holt 0 Blöcke auf)} \\
&+ \ldots \\
&+ \text{P(Angreifer hat k Blöcke erstellt)} \cdot \text{P(Angreifer holt } z-k \text{ Blöcke auf)}
\end{aligned}
$$

Der Fortschritt des Angreifers ist Poisson verteilt mit einer Blockrate $\lambda = z\frac{q}{p}$ und ergibt sich über:

$$P(\text{Angreifer hat k Blöcke erstellt}) = P_\lambda(k) = \frac{\lambda^k}{k!}e^{-\lambda}$$

Die Wahrscheinlichkeit, dass der Angreifer $z-k$ Blöcke aufholt, kann von q_z abgeleitet werden:

$$P(\text{Angreifer holt } z-k \text{ Blöcke auf}) = \begin{cases} 1 & \text{für } k \geq z \\ \left(\frac{q}{p}\right)^{z-k} & \text{für } k < z \end{cases}$$

Daraus ergibt sich folgende Gleichung:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} 1 & \text{für } k \geq z \\ \left(\frac{q}{p}\right)^{z-k} & \text{für } k < z \end{cases}$$

Nimmt man nun das Gegenereignis „Angreifer kann den Abstand nicht mehr aufholen“ und subtrahiert dessen Wahrscheinlichkeit von 1, so kann die unendliche Summe vermieden werden und man bekommt folgende Gleichung:

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \left(1 - \left(\frac{q}{p}\right)^{z-k}\right)$$

Nimmt der Empfänger an, dass der Angreifer 10 % der gesamten Rechenleistung besitzt ($q = 0,1$), ergibt sich eine Erfolgschance von lediglich 0,35 % für den Angriff.
In Abbildung 7.2 sind die Erfolgschancen eines Angreifers für verschiedene Werte von q eingezeichnet.
Obwohl die 51 %-Attacke ihren Namen von der erforderlichen Mehrheit der Rechenleistung ableitet, kann sie theoretisch auch mit weniger als 50 % der Rechenleistung erfolgreich sein. In solchen Fällen ist der Angreifer jedoch auf Glück angewiesen, da er vorübergehend mehr Blöcke als das restliche Netzwerk erzeugen muss.
Solche Angriffe sind theoretisch möglich, wurden jedoch bislang noch nicht beobachtet. Ein möglicher Grund dafür ist, dass es für Schürfer wirtschaftlich sinnvoller ist, „ehrlich“ zu bleiben und die Blockbelohnung zu kassieren. Ein erfolgreicher Angriff könnte das

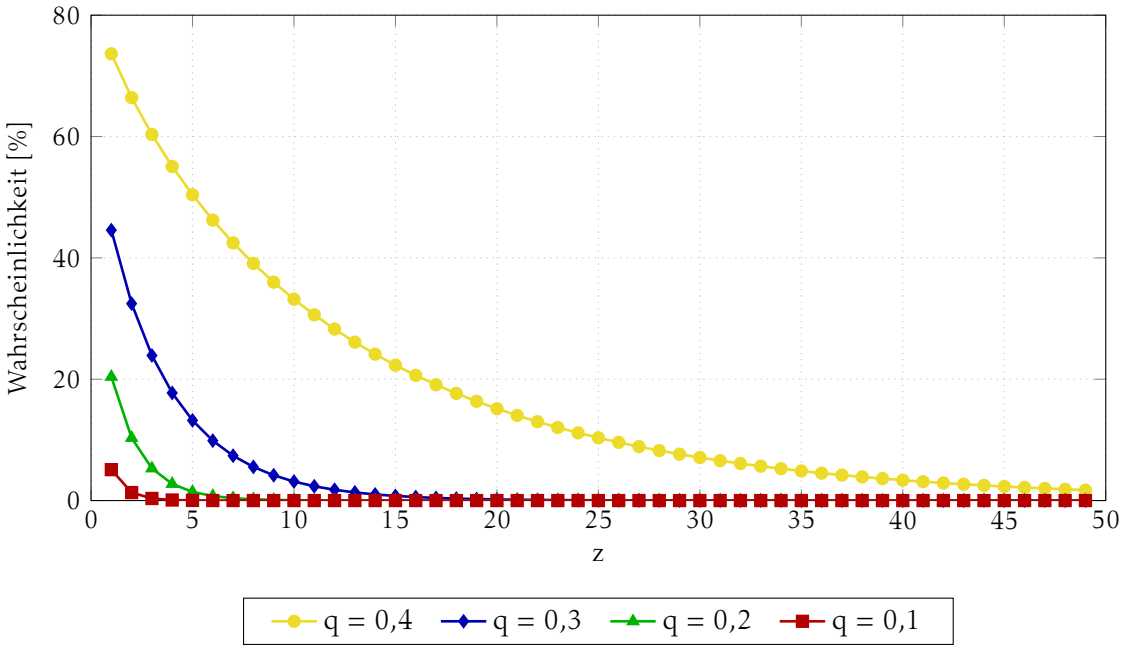


Abbildung 7.2: *Erfolgschancen eines Angreifers*

Vertrauen in Bitcoin und seine Akzeptanz in der Gesellschaft beeinträchtigen. Dies hätte negative Auswirkungen auf den Bitcoin-Kurs, wodurch die vom Angreifer erbeuteten Bitcoins an Wert verlieren würden. Angesichts dieser Risiken bleibt der Anreiz für einen solchen Angriff gering.

8 Fazit

8.1 Zusammenfassung

Im Gegensatz zum traditionellen Bankensystem erfordert Bitcoin kein Vertrauen in eine zentrale Institution. Ebenso ist kein Vertrauen in die Teilnehmer des Bitcoin-Netzwerks notwendig. Es muss lediglich dem Bitcoin-Protokoll und den zugrunde liegenden kryptografischen Verfahren vertraut werden. In dieser Arbeit wurden die wesentlichen technischen Aspekte von Bitcoin analysiert und anschaulich erklärt.
Es wurde dargelegt, wie Hashfunktionen aufgrund ihrer kryptografischen Eigenschaften die Datenintegrität der Blockchain sicherstellen und wie sie in einer Baumstruktur eine effiziente Verwaltung der Transaktionen ermöglichen. Darüber hinaus wurde der Elliptic Curve Digital Signature Algorithm (ECDSA) beschrieben, der zur Erstellung und Verifikation von Signaturen dient.
Der grundlegende Aufbau von Bitcoin-Transaktionen wurde erklärt. Es wurde beschrieben, wie neue Bitcoins durch die speziellen Coinbase-Transaktionen generiert werden und wie die maximale Anzahl an Bitcoins begrenzt ist. Die Bitcoin-eigene Skriptsprache wurde vorgestellt, die mithilfe der beschriebenen Schlüssel und Signaturen den Zugriff auf die Bitcoins absichert. Dabei wurden die beiden Standard-Skripte P2PK und P2PKH detailliert erklärt. Darüber hinaus wurden Bitcoin-Adressen eingeführt, die es ermöglichen, ein komplexes Standard-Skript in eine benutzerfreundliche Darstellung umzuwandeln. Dadurch ähnelt ein Skript einer Kontonummer, an die Bitcoins einfach gesendet werden können.
Der Kern von Bitcoin ist die Blockchain, eine verteilte Datenbank, die alle Transaktionen in einer Kette von Blöcken speichert. Diese innovative Datenstruktur, in Kombination mit den beschriebenen kryptografischen Verfahren, ermöglicht es den Teilnehmern des genehmigungslosen P2P-Netzwerks von Bitcoin, Transaktionen ohne gegenseitiges Vertrauen durchzuführen. Die Miner erfüllen im Bitcoin-Netzwerk die Aufgabe, die Blockchain durch das Hinzufügen neuer Blöcke zu erweitern. Ihr Anreiz liegt in der Vergabe von Blockbelohnungen für das erfolgreiche Erstellen eines Blockes. Neben den Minern gibt es weitere Netzwerk-Teilnehmer, deren Hauptaufgabe die Validierung von Transaktions- und Blockdaten anhand der Regeln des Bitcoin-Protokolls ist.
Abschließend wurde ein potenzielles Angriffsszenario auf die Blockchain analysiert, wobei deutlich wurde, dass die Sicherheit eines Blocks mit wachsender Blocktiefe exponentiell zunimmt.
Zusammenfassend bietet diese Arbeit einen umfassenden Überblick über die technischen Grundlagen von Bitcoin und zeigt, wie durch diese ein digitales, sicheres und dezentrales Zahlungssystem geschaffen wurde.

8.2 Ausblick

Diese Arbeit konzentrierte sich auf die grundlegende technische Funktionsweise von Bitcoin. Darüber hinaus gibt es jedoch zahlreiche weitere Themenbereiche, die Potenzial für eine weiterführende Untersuchung bieten. Nachfolgend werden einige dieser Bereiche vorgestellt.

Detaillierte Analyse der Bitcoingrundlagen

Einige Abschnitte dieser Arbeit wurden bewusst vereinfacht dargestellt, um das Verständnis zu erleichtern, wie beispielsweise der Aufbau einer Transaktion. Eine genauere Untersuchung könnte sich mit der tatsächlichen Implementierung des Bitcoin-Protkolls befassen. Zudem wurden die in Bitcoin verwendeten Skripttypen wie P2SH und P2WPKH lediglich erwähnt und könnten ausführlicher behandelt werden. Darüber hinaus existieren weitere Standard-Skripte, die bislang unbesprochen blieben.

Aktualisierung des Protokolls

Änderungen am Bitcoin-Protokoll erfolgen über sogenannte Forks. Diese können entweder als Soft Forks, die abwärtskompatibel sind, oder als Hard Forks, die inkompatible Änderungen einführen, umgesetzt werden. Eine detaillierte Untersuchung könnte die beiden Fork-Typen vergleichen und vergangene Protokoll-Aktualisierungen analysieren. Zudem könnte sich eine weitergehende Arbeit damit beschäftigen, wie die Teilnehmer des Netzwerks sich einigen, ob eine Aktualisierung durchgeführt werden soll oder nicht. Dies bietet einen Einblick in die Entscheidungsmechanismen des Netzwerkes.

Weitere Angriffsszenarien

Neben der in dieser Arbeit behandelten 51 %-Attacke existieren weitere potenzielle Angriffsszenarien, die das Bitcoin-Netzwerks gefährden könnten. Eine umfassende Analyse solcher Szenarien könnte wertvolle Erkenntnisse liefern.

Wallets

Wallets spielen eine zentrale Rolle in der Verwaltung von Bitcoins, wurden jedoch in dieser Arbeit nicht näher beschrieben. Eine Betrachtung der verschiedenen Wallet-Typen und ihrer Funktionsweise könnte ein tieferes Verständnis der Bitcoin Verwaltung ermöglichen.

Skalierbarkeit von Bitcoin

Da Transaktionen in Bitcoin mit Gebühren verbunden sind und neue Blöcke lediglich alle Zehn Minuten erstellt werden, ist das System für eine hohe Anzahl kleiner Transaktionen wenig geeignet. Das auf Bitcoin aufbauende Lightning Network bietet eine Lösung durch schnellere Zahlungen und geringere Gebühren. Eine genauere Analyse dieses Netzwerks könnte aufzeigen, wie Lösungen wie das Lightning Network die Skalierbarkeit von Bitcoin verbessern und dazu beitragen, Bitcoin vermehrt in den Alltag als Zahlungsmittel zu integrieren.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] J. D. Stein, *Cosmic numbers: the numbers that define our universe*, en. New York: Basic Books, 2011, ISBN: 978-0-465-02198-7.

[2] O. R. L. C. Facility, *Frontier*, englisch. Adresse: https://www.olcf.ornl.gov/frontier/ (besucht am 05. 11. 2024).

[3] L. Harald, *Wie wird das Alter des Universums berechnet?* de, Nov. 2022. Adresse: https://www.swr.de/wissen/1000-antworten/wie-wird-das-alter-des-universums-berechnet-100.html (besucht am 05. 11. 2024).

[4] National Institute of Standards and Technology (US), „SHA-3 standard : permutation-based hash and extendable-output functions," en, National Institute of Standards und Technology (U.S.), Washington, D.C., Techn. Ber. error: 202, 2015. DOI: 10.6028/NIST.FIPS.202. Adresse: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf (besucht am 09. 10. 2024).

[5] C. Paar und J. Pelzl, *Kryptografie verständlich* (eXamen.press), de. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, ISBN: 978-3-662-49296-3 978-3-662-49297-0. DOI: 10.1007/978-3-662-49297-0. (besucht am 07. 10. 2024).

[6] W. Ertel und E. Löhmann, *Angewandte Kryptographie*, de, 6., aktualisierte Auflage. München: Hanser, 2020, ISBN: 978-3-446-46313-4.

[7] National Institute of Standards and Technology (US), „Secure hash standard," en, National Institute of Standards und Technology (U.S.), Washington, D.C., Techn. Ber. NIST FIPS 180-4, 2015. DOI: 10.6028/NIST.FIPS.180-4. Adresse: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf (besucht am 04. 11. 2024).

[8] H. Dobbertin, A. Bosselaers und B. Preneel, „RIPEMD-160: A strengthened version of RIPEMD," en, in *Fast Software Encryption*, G. Goos, J. Hartmanis, J. Leeuwen und D. Gollmann, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, ISBN: 978-3-540-60865-3 978-3-540-49652-6. DOI: 10.1007/3-540-60865-6_44. (besucht am 04. 11. 2024).

[9] R. C. Merkle, „A Digital Signature Based on a Conventional Encryption Function," in *Advances in Cryptology — CRYPTO '87*, C. Pomerance, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, ISBN: 978-3-540-48184-3.

[10] G. Walker, *Merkle Root | A Fingerprint for the Transactions in a Block*, en, Sep. 2024. Adresse: https://learnmeabitcoin.com/technical/block/merkle-root/#merkle-tree (besucht am 20. 11. 2024).

[11] Standards for Efficient Cryptography, „SEC 1: Elliptic Curve Cryptography," en, Nr. 2, Mai 2009. (besucht am 08. 11. 2024).

[12] National Institute of Standards and Technology (US), „Digital Signature Standard (DSS),“ en, National Institute of Standards und Technology (U.S.), Washington, D.C., Techn. Ber. NIST FIPS 186-5, Feb. 2023. DOI: 10.6028/NIST.FIPS.186-5. Adresse: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf (besucht am 07. 11. 2024).

[13] Standards for Efficient Cryptography, „SEC 2: Recommended Elliptic Curve Domain Parameters,“ en,

[14] *Elliptic Curve Plotter*. Adresse: https://kebekus.gitlab.io/ellipticcurve/ (besucht am 23. 12. 2024).

[15] S. Nakamoto, „Bitcoin: A Peer-to-Peer Electronic Cash System,“ en, Mai 2009.

[16] *Script - Bitcoin Wiki*, en. Adresse: https://en.bitcoin.it/wiki/Script (besucht am 21. 12. 2024).

[17] A. M. Antonopoulos, *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*, en. O'Reilly Media, 2014, ISBN: 978-1-4919-2198-2.

[18] G. Walker, *Technical Guide | How To Become a Bitcoin Programmer*, en, Nov. 2024. Adresse: https://learnmeabitcoin.com/technical/ (besucht am 10. 12. 2024).

[19] *Blockchain_Eigenschaften*, de, Sep. 2024. Adresse: https://de.wikipedia.org/w/index.php?title=Blockchain&oldid=248611873#Eigenschaften (besucht am 12. 12. 2024).

[20] S. Haber und W. S. Stornetta, „How to time-stamp a digital document,“ en, *Journal of Cryptology*, Nr. 2, 1991.

[21] *Protocol rules - Bitcoin Wiki*. Adresse: https://en.bitcoin.it/wiki/Protocol_rules (besucht am 30. 12. 2024).

Glossar

$\mathbb{F}_p$ Endliches Primfeld bestehend aus p Elementen, wobei p eine Primzahl ist.

ANSI American National Standards Institute.

ECC elliptische Kurven Kryptographie.
ECDSA Elliptic Curve Digital Signature Algorithm.
EF = ExaFLOPS = $1,6 \times 10^{18}$ FLOPS (Floating Point Operations Per Second), also 1,6 Trillionen Fließkomma-Operationen pro Sekunde.

FLOPS Floating Point Operations Per Second.

HD Wallets Hierarchical Deterministic Wallets.

Mempool Memory Pool.

NIST National Institute of Standards and Technology.
Nonce Number used only once.

P2P peer-to-peer.
P2PK Pay To Public Key.
P2PKH Pay To Public Key Hash.
P2SH Pay To Script Hash.
P2WPKH Pay To Witness Public Key Hash.
P2WSH Pay To Witness Script Hash.
PBKDF2 Password Based Key Derivation Function 2.

RACE **R**esearch and Development in **A**dvanced **C**ommunications Technologies in **E**urope.
RNG Random Number Generator.

SHA Secure Hash Algorithm.
SHS Secure Hash Standard.
SPV Simplified Payment Verification.

TxID Transaktions ID.

UTxO Unspent Transaction Output.