UNIVERSITÄT KONSTANZ

Hashfunktion und Blockchain

Fachseminar Zahlentheorie
Prof. Dr. Salma Kuhlmann

14. Juli 2021

Autor: Rebecca Köchling Mayán
Betreuer: Michele Serra

Abstract

In diesem Vortrag geht es um Hashfunktionen. Zuerst werden Grundlagen aus der Informationstheorie eingeführt. Mittels Modularer Hashfunktionen wird die Bedeutung des Aufbaus von Hashfunktionen veranschaulicht und bedeutende Merkmale zur Kollisionsreduktion aufgezeigt. Der Schwerpunkt des Vortrags liegt auf kryptografischen Hashfunktionen. Dazu werden elementare Sicherheitsanforderungen wie die Kollisionsresistenz und Urbildresistenz betrachtet. Infolgedessen wird mittels des Merkle-Meta-Verfahrens eine zentrale Konstruktion von kollisionsresistenten Hashfunktionen vorgestellt. Schließlich wird die Sicherheit der digitalen Signatur in Kombination mit Hashfunktion verdeutlicht. Den Abschluss bildet die Vorstellung der Blockchain Technologie und dessen fundamentale Anwendung von Hashfunktionen in der Blockchain.

Inhaltsverzeichnis

1 Hashfunktionen

Die zunehmende Digitalisierung führt zu einem exorbitanten Anstieg von Datenmengen, die in lokalen und weltweiten Datennetzen verarbeitet werden. Da das Suchen in großen Datenmengen ein sehr ressourcenintensives Verfahren ist werden Hashfunktionen genutzt, die einem beliebig langen Originaltext einen Text kürzerer, fester Länge zuordnen. Für jedes Datenobjekt einer Datenmenge wird ein sogenannter Hashwert berechnet, der den Eintrag eindeutig identifiziert. Sucht man nun nach einem Datenobjekt, so wird aus dem Suchbegriff wieder ein Hashwert errechnet, der mit den vorliegenden Hashwerten verglichen wird. Dieses Verfahren ist wesentlich effizienter da nur die kurzen Hahswerte verglichen werden müssen und nicht die komplette Datenbank. Um für den Computer verarbeitbar zu sein, müssen alle Informationen zuerst in Zahlen umgewandelt werden. Für diese Kodierung von alphanumerischen Zeichen wird der ASCII [1] verwendet, eine 7 – Bit – Zeichenkodierung, welche 128 Zeichen definiert.

1.1 Definition und Grundlagen

Defintion 1.1 (String)**.** Ein String ist in der Informationstheorie eine endliche Folge von Zeichen $(z_i)_{i=1,\dots,n}$ (z.B. Ziffern) aus einem definierten Zeichensatz $Z := \{z_i | i = 1, \dots, n\}$.

Defintion 1.2 (Bit)**.** Ein Bit (engl. *binary digit*) ist die Bezeichnung für eine Stelle einer Binärzahl (0 und 1). Bitfolgen sind endliche Zeichenketten, die mit den Elementen 0 und 1 der Menge $\mathbb{B} = \{0, 1\}$ gebildet werden können. Die Länge einer Bitfolge n ist die Anzahl der in ihr vorkommenden Binärzahlen, also ist $|\mathbb{B}^n| = 2^n$.

Defintion 1.3 (Alphabet)**.** Sei $\sum$ eine endliche, nichtleere Menge von Zeichen. So ist

$$\sum^{m} = \left\{ a_1 a_2 \cdots a_m : a_i \in \sum \right\}$$

die Menge der Wörter, Strings, Zeichenketten der Länge m und

$$\sum^{*} = \left\{ a_1 a_2 \cdots a_m : a_i \in \sum, m \geq 0 \right\} = \bigcup_{m \geq 0} \sum^{m}$$

die Menge aller möglichen Wörter, Strings oder Zeichenketten. $\sum$ heißt Alphabet.

[1] **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange

Für diese Arbeit sei $\sum := \{0,1\}$ das binäre Alphabet der Bits, $\sum^n := \{0,1\}^n$ die Menge aller Bitstrings der Länge n und $\sum^* := \{0,1\}^*$ die Menge aller endlichen Bitstrings.

Defintion 1.4 (Kompressionsfunktion)**.** Eine Kompressionsfunktion ist eine Abbildung

$$f : \sum^m \to \sum^n \qquad m, n \in \mathbb{N}, m > n$$

Sie bildet Bitstrings einer festen Länge m auf Bitstrings fester kürzerer Länge n ab. [3, S. 234]

Defintion 1.5 (Hashfunktion)**.** Sei $n > 0$. Eine Funktion

$$h : \sum^* \to \sum^n$$

heißt *Hashfunktion* (Streuwertfunktion). Die Zahl n ist die *Hashlänge*. Der Bitvektor $h(m)$ wird *Hashwert* von $m \in \sum^*$ genannt.

Da Hashfunktionen beliebig lange Strings auf Strings fest vorgegebener Länge n abbilden, sind sie nicht injektiv [2] [6, S. 119]. Das heißt für verschiedene Eingaben treten identische Hashwerte auf. Dies wird Kollision genannt.

Defintion 1.6 (Kollision)**.** Eine *Kollision* von h ist ein Paar $(m, m') \in \{0,1\}^2$ von Strings, für die $m \neq m'$ und $h(m) = h(m')$ gilt.

1.2 Modulare Hashfunktionen

1.2.1 Divisions – Rest – Methode

Definition 2.1. Seien die Bitstrings natürliche Zahlen, das heißt sei $m \in \mathbb{N}$.
Die *Divisions – Rest – Methode* liefert eine Hashfunktion

$$h : \sum^* \longrightarrow \{0, ..., p-1\}, \qquad m \;\mapsto\; m \pmod{p}\,.$$

Bemerkung 2.2. Die Wahl für $p = 2^i$ ist hier ungeeignet, da dies der Extraktion der i-niedrigstwertigen Bits von m entspricht. Wähle optimalerweise eine Primzahl für p. [2]

[2] Eine Abbildung $f : X \longrightarrow Y$ heißt *injektiv*, falls aus $f(x) = f(y)$ folgt $x = y$.

1.2.2 Multiplikative Methode

Definition 2.3. Sei $m \in \mathbb{N}$ und $c \in \mathbb{R}$, $0 < c < 1$. Die *Multiplikative Methode* liefert eine Hashfunktion

$$h : \sum^{*} \longrightarrow \{0, ..., p-1\}, \;\; m \mapsto \lfloor p \cdot (m \cdot c - \lfloor m \cdot c \rfloor) \rfloor,$$

wobei die Gaußklammer $\lfloor x \rfloor := \max\{k \in \mathbb{Z} | k \leq x\}$ ist. [6, S. 113]

Bemerkung 2.4. Entscheidend für die Verteilung von m auf den Wertebereich ist die Wahl von c, welche unabhängig von der Wahl von m ist. Idealerweise wählt man für c den goldenen Schnitt, d.h. $c = \phi^{-1} = \frac{\sqrt{5}-1}{2}$ [6, S. 113]. Somit ist die Verteilung aufgrund der Eigenschaft der irrationalen Zahlen nahezu gleich verteilt. Wird der goldene Schnitt verwendet, so nennt man die resultierende Hashfunktion auch den Fibonacci – Hash.

1.3 Kryptografische Hashfunktionen

Hashfunktion sind ein kryptografisches Primitiv, sie prüfen Nachrichten auf Integrität. Bei der Übermittlung von Daten soll der Empfänger feststellen können, ob die Daten oder die Nachricht nach ihrer Erzeugung verändert wurden. Sie werden in vielen kryptografischen Verfahren eingesetzt und sind beispielsweise ein wichtiger Teil von Signaturverfahren. Die Resistenz gegen Angriffe auf kryptografische Verfahren hängt stark von den in diesen Algorithmen genutzten Hashfunktionen ab. Deshalb stellt man hohe Sicherheitsanforderungen an diese Hashfunktionen.

1.3.1 Sicherheitseigenschaften

- Effizienz: Hashfunktionen sollen eine Nachricht beliebiger Länge verarbeiten können. Von daher sollen sie sehr schnell und einfach zu berechnen sein.
- Der Ausgangswert soll eine feste Länge haben, der unabhängig von der Länge des Eingangs ist.
- Repräsentativität: Der gesamte berechnete Hashwert soll von allen Eingangsbits abhängen. Daraus folgt, dass selbst kleine Änderungen im Eingangswert zu einem vollständig unterschiedlichen Hashwert führen sollen.

Beispiel 3.1. Eingesetzt in SHA – 256[3] bekommt man folgende Hashwerte:

Text-Datei von Goethes Faust (187.937 Zeichen)

↓

812927030cefaf404bfd9879f06b46faab9fdea4a11a5f52de8a03b945c9392f

Zahlentheorie

↓

c96bf2832d9e9a3cc6e3c752f007ce4b61ca624c55e45e6008c176be51c9d7a4

Zahlentheori

↓

921cd7c8bcc185c1c61e2fe49dbbe00ce9d709862a665b2e4987eb4c6acf9554

Defintion 3.2 (Kryptografische Hashfunktion)**.**

(i) Eine Hashfunktion h heißt *schwach kollisionsresistent,* wenn es keinen effizienten Algorithmus gibt, der bei Eingabe von m ein m' findet mit $h(m) = h(m')$

(ii) Eine Hashfunktion h heißt *stark kollisionsresistent*, wenn es nicht effizient möglich ist, zwei verschiedene Bitstrings m und m' mit $m \neq m'$ zu finden, für die $h(m) = h(m')$ gilt.

(iii) Eine Hashfunktion h ist eine *Einweg-Hashfunktion*, wenn es nicht effizient möglich ist, zu einem zufällig gewählten Hashwert y einen Bitstring m mit $h(m) = y$ zu finden. Diese Eigenschaft wird auch *Urbildresistenz* genannt.

Erfüllt eine Hashfunktion diese Eigenschaften, so nennt man sie eine *Kryptografische Hashfunktion.* [1, S. 177]

1.3.2 Exkurs: Komplexitätstheorie

Die Komplexitätstheorie befasst sich mit der Theorie von Algorithmenlaufzeiten. Polynomielle Laufzeit [4] garantiert, dass sich ein Algorithmus relativ schnell durchführen lässt. Eines der wichtigsten und ungelösten Probleme ist das P – NP – Problem: Es ist ungeklärt,

[3]Der **S**ecure **H**ash **A**lgorithm – 256 besteht aus Folgen von 256 Bits, dargestellt als Folge von 64 Hexadezimalzahlen. (Das Hexadezimalsystem ist ein System zur Basis $b = 16$ bestehend aus $(0, \ldots, 9, a, \ldots f)$ wobei $a = 10$, $b = 11$, ... entspicht.)

[4]Die *Laufzeit* L eines Algorithmus ist die Anzahl von Operationen, die vom Algorithmus bis zur Ausgabe durchgeführt werden. Lässt sie sich durch ein Polynom in der Eingabelänge nach oben abschätzen (gilt also: $L(n) = O(P(n))$ für ein Polynom P), so heißt der Algorithmus *polynomiell* . [1, S. 46]

ob P $\neq$ NP gilt. Die Klasse P beinhaltet Probleme, zu deren Lösung es Algorithmen mit polynomieller Laufzeit gibt. Und die Klasse NP fasst alle Probleme zusammen, bei denen der Aufwand zur Verifikation einer Lösung polynomiell ist. NP – vollständige Probleme lassen sich nichtdetermnistisch in Polynomialzeit lösen und gelten als praktisch unlösbar. Findet man für ein einziges NP – vollständiges Problem einen Lösungsalgorithmus mit polynomieller Laufzeit, so folgt daraus bereits, dass alle Probleme in NP in polynomieller Zeit lösbar sind und damit gilt P = NP. [8, vgl. Kap. 4]

Eine Einwegfunktion ist komplexitätstheoreisch leicht berechenbar, aber schwer umzukehren. Ihre Existenz ist nicht bewiesen [1, S. 180]. Tatsächlich würde der Beweis ihrer Existenz gleichzeitig den Beweis für P $\neq$ NP bedeuten[5]. Aus diesem Grund geben wir nur eine Formale Definition.

"Defintion 3.3" (Einwegfunktion)**.** Eine Funktion $h : \sum^* \to \sum^n$ heißt *Einwegfunktion*, wenn es praktisch unmöglich ist zu einem $y \in \sum^n$ ein $m \in \sum^*$ mit $h(m) = y$ zu finden.

Obwohl die starke Kollisionsresistenz die Einweg-Eigenschaft nicht impliziert [1, S. 178], ist sie gewissermaßen eine stärkere Eigenschaft als die Einweg – Eigenschaft. Dies veranschaulicht der folgende Satz, der den Aufwand abschätzt eine Kollision zu finden.

Satz 3.4 (Kollisionsabschätzung)**.** Es sei $h : \sum^* \to \sum^n$ eine kryptografische Hashfunktion und $h(m)$ sei gleichmäßig und unabhängig auf $\{0, 1, ..., 2^n - 1\}$ verteilt. Wenn ein Angreifer mehr als $1,2 \cdot 2^{\frac{n}{2}}$ mal zufällig ein $m \in \sum^*$ ausgewählt hat, so hat er mit einer Wahrscheinlichkeit von mindestens $\frac{1}{2}$ ein m und m' gefunden mit $h(m) = h(m')$. [1, S. 179]

Beweis: Wegen $|\sum^n| = 2^n$ ist klar, dass mindestens 2^n Kollisionen existieren.
Die Wahrscheinlichkeit, dass nach k Ziehungen keine Kollision stattfindet ist:

$$p_{kk} = \left(\frac{2^n}{2^n}\right)\left(\frac{2^n-1}{2^n}\right)\cdots\left(\frac{2^n-(k-1)}{2^n}\right) = \prod_{i=0}^{k-1}\left(\frac{2^n-i}{2^n}\right) = \prod_{i=1}^{k-1}\left(1-\frac{i}{2^n}\right)$$

Da für alle $x \in \mathbb{R}$ gilt: $1 - x \leq e^{-x}$, folgt

$$p_{kk} = \prod_{i=1}^{k-1}(1-\frac{i}{2^n}) \leq \prod_{i=1}^{k-1} e^{-\frac{i}{2^n}} = e^{-\frac{1}{2^n}\sum_{i=1}^{k-1} i} = e^{-\frac{k(k-1)}{2\cdot 2^n}}$$

[5]Aus P $\neq$ NP folgt jedoch nicht die Existenz von Einwegfunktionen.

Somit ist die Wahrscheinlichkeit $p_{koll} = 1 - p_{kk}$ für mindestens eine Kollision

$$p_{koll} \leq 1 - e^{-\frac{k(k-1)}{2 \cdot 2^n}} \qquad \text{bzw.} \qquad e^{-\frac{k(k-1)}{2 \cdot 2n}} \leq 1 - p_{koll}.$$

Durch Logarithmusbildung und Umformung folgt

$$-\frac{k(k-1)}{2 \cdot 2^n} \leq \ln(1 - p_{koll}) \qquad \text{bzw.} \qquad \frac{k(k-1)}{2 \cdot 2^n} \geq \ln\left(\frac{1}{1 - p_{koll}}\right)$$

und damit

$$k(k-1) = k^2 - k = \left(k - \frac{1}{2}\right)^2 - \frac{1}{4} \geq 2 \cdot 2^n \ln\left(\frac{1}{1 - p_{koll}}\right) \leq$$

Also ist

$$k \geq \frac{1}{2} + \sqrt{\frac{1}{4} + 2 \cdot 2^n \ln\left(\frac{1}{1 - p_{koll}}\right)} > \sqrt{2 \cdot \ln\left(\frac{1}{1 - p_{koll}}\right)} \cdot \sqrt{2^n}.$$

Wählen wir nun $p_{koll} = \frac{1}{2}$ ergibt sich die Schätzung

$$k \geq \sqrt{2 \cdot \ln(2)} \cdot 2^{\frac{n}{2}} \approx 1,2 \cdot 2^{\frac{n}{2}}.$$

Das bedeutet, dass man mit einer Wahrscheinlichkeit von $\frac{1}{2}$ bei ungefähr $\sqrt{2^n}$ zufällig gewählten Elementen aus $\sum^*$ eine Kollision erhält, während man zum Auffinden eines Urbildes 2^{n-1} Elemente testen muss. [1, S.178] □

Bemerkung 3.5. Aufgrund dieses Phänomens (auch als Geburtstagsparadoxon bekannt) ist es deutlich leichter, eine beliebige Kollisionen zu finden als ein Urbild. Um Angriffe die dieses Phänomen nutzen, sogenannte Geburtstagsattacken, zu verhindern, muss die Hashwertlänge also $n \geq 128$ Bit[6] sein. Deswegen entspricht die Ausgabelänge der meisten Hashfunktionen der doppelten Länge des gewünschten Sicherheitslevels: Soll eine Hashfunktion etwa 128 Bit Sicherheit gegen Kollisionen bieten, was 2^{128} Rechenoperationen entsprechen, so benötigt sie eine Ausgabe von 256 Bit ($\sqrt{2^{256}} = 2^{128}$).

Somit ist die Länge n des zurückgegebenen Hashwertes der wichtigste Sicherheitsparameter, bei kryptografischen Hashfunktionen.

[6]Die in den BSI empfohlenen Hashfunktinen haben eine Mindestlänge von 256 Bits. Für den Vorhersagezeitraum nach Ende 2022 wird die Verwendung von Verfahren empfohlen, die ein Sicherheitsniveau von mindestens 120 Bit erreichen. [4]

1.3.3 Merkle – Damgård – Konstruktion

Das Merkle – Meta –Verfahren[7] beschreibt eine Methode zur Konstruktion von kryptografischen Hashfunktionen. Dabei wird eine Kompressionsfunktion so iteriert, dass Nachrichten beliebiger Länge verarbeitet werden können. Sie dient als Basis fast aller in der Praxis eingesetzten Hashfunktionen.

Satz 3.6 (Merkle – Meta – Verfahren)**.** Sei

$$f : \{0,1\}^m \to \{0,1\}^n, \qquad r = m - n \text{ wobei } r > 0$$

eine kollisionsresistente Kompressionsfunktion. Wird das Merkle – Meta – Verfahren auf f angewendet, so ist die resultierende Hashfunktion

$$h : \{0,1\}^* \to \{0,1\}^n$$

kollisionsresistentent. [3, S. 237]

Wir zeigen die Konstruktion für $r \geq 2$. Sei also $m \in \{0,1\}^*$.

Die eingegebene Nachricht m wird mit dem sogenannten *Padding – Verfahren* zuerst in Blöcke fester Länge geteilt und mit zusätzlichen Bits aufgefüllt, so dass $\overline{m}$ ein ganzzahliges Vielfaches der Blocklänge r ist. Dabei geht man folgendermaßen vor: Zuerst schreiben wir vor m eine minimale Anzahl an Nullen, sodass die neue Länge durch r teilbar ist, und hängen an diesen String nochmal r Nullen. Jetzt bestimmen wir die Binärentwicklung der Länge des originalen Strings m. Dieser Binärentwicklung stellen wir so viele Nullen voran, sodass die Länge durch $r-1$ teilbar ist. Vor jedes $(r-1)\cdot j$-te Zeichen $(j = 1, 2, ...)$ dieser Binärentwicklung wird eine 1 geschrieben. Diesen neuen String hängen wir nun an den vorherigen String. Und zuletzt wird der Gesamtstring in t Blöcke der Länge r aufgeteilt und mit der Konkatenation [8] $\circ$ verbunden zu

$$\overline{m} = m_1 \ \circ \ m_2 \ \circ \ ... \ \circ \ m_t \qquad \text{mit} \qquad m_i \in \{0,1\}^r, \qquad 1 \leq i \leq t.$$

[7] Das Verfahren basiert auf Arbeiten von Ralph C. Merkle und Ivan Bjerre Damgård.

[8] Die *Konkatenation* oder *Verkettung* ist eine Verknüpfung zweier Wörter zu einem neuen Wort, das durch Aneinanderhängen der beiden Symbolfolgen entsteht. Die Konkatenation der beiden Wörter x und y über einem Alphabet $\sum$ wird mit xy oder $x \circ y$ angegeben und ist definiert durch $xy = x \circ y := (x_1, x_2, x_3, \ldots, x_n, y_1, y_2, y_3, \ldots, y_k)$.

Beispiel 3.7. Für $m = 1110100101$ und $r = 4$ ist $\overline{m} = \underbrace{0011}_{m_1} \circ \underbrace{1010}_{m_2} \circ \underbrace{0101}_{m_3} \circ \underbrace{1001}_{m_4} \circ \underbrace{1010}_{m_5}$.

Wir berechnen den Hashwert $h(m)$ iterativ:

Für den 1. Block wird ein Initialwert $IW =: h_0$ verwendet. Wir setzen $h_0 = 0^n$. Sukzessiv wird nun jeder Block mit dem bisherigem Zwischenwert verknüpft mit

$$h_i := f(h_{i-1} \circ m_i) \qquad 1 \leq i \leq t.$$

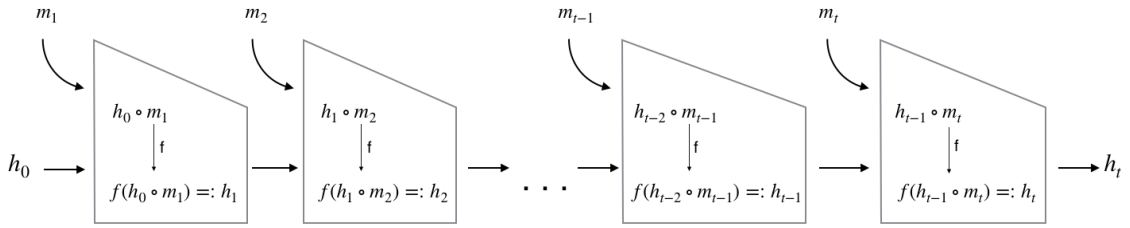

Abbildung 1: Merkle – Damgård – Konstruktion

Die Kompressionsfunktion hat somit als Input einen Nachrichtenblock und den Output der vorherigen Nachrichtenblöcke. Der Hashwert des letzten Blocks $h_t = h(m)$ ist der Hashwert der gesamten Nachricht.

Nun zeigen wir, dass wenn f kollisionsresistent ist, daraus folgt, dass auch h kollisionsresistent ist. Da wir die Kollisionsresistenz nicht formal definiert haben, geben wir hier keinen expliziten Beweis sondern nur eine Beweisidee an.

Beweisskizze: Wir zeigen, dass man aus einer Kollision von h eine Kollision von f bestimmen kann (vgl. [3, S. 237 ff.]).
Sei also (m, m') eine Kollision von h und $m_1, ..., m_t, m'_1, ..., m'_{t'}$ die zugehören Strings der Länge r. Die entsprechen Strings von Hashwerten seien $h_0, ..., h_t, h'_0, ..., h'_{t'}$.
Weil (m, m') eine Kollision ist, gilt

$$h_t = h'_{t'}.$$

Sei $t \leq t'$. Wir vergleichen nun

$$h_{t-i} \quad \text{mit} \quad h'_{t'-i} \qquad \text{für} \quad i = 1, ..., t-1.$$

Angenommen, wir finden ein $i < t$ mit

$$h_{t-i} = h'_{t'-i} \qquad \text{und} \qquad h_{t-i-1} \neq h'_{t'-i-1},$$

dann gilt

$$h_{t-i-1} \ \circ \ m_{t-i} \neq h'_{t'-i-1} \ \circ \ m'_{t'-i}$$

und durch Anwenden von f

$$f(h_{t-i-1} \ \circ \ m_{t-i}) = h_{t-i} = h'_{t'-i} = f(h'_{t'-i-1} \ \circ \ m'_{t'-i}).$$

Dies ist eine Kollision von f.
Sei nun angenommen, dass

$$h_{t-i} = h'_{t'-i} \qquad \text{für } \ i = 1 \ldots t.$$

Wir zeigen nun, dass es einen Index i gibt, mit

$$m_{t-i} \neq m'_{t'-i} \qquad \text{und } \ i = 0 \ldots t-1. \tag{*}$$

Fall 1: Sind die Längen von m und m' verschieden, und werden für die Darstellung der Länge von m weniger Strings gebraucht als für die Darstellung der Länge von m', dann gibt es einen Index i, für den m_{t-i} nur aus Nullen besteht und für den $m'_{t'-i}$ eine führende 1 enthält. Werden für die Darstellung der Längen von m und m' gleich viele Strings gebraucht, dann gibt es einen Index i derart, dass m_{t-i} und $m'_{t'-i}$ in der Darstellung der Länge von m bzw. m' vorkommen und verschieden sind.

Fall 2: Sind die Längen von m und m' gleich, dann gibt es einen Index i mit $m_{t-i} \neq m'_{t'-i}$, weil m und m' verschieden sind.

Da wir (*) gezeigt haben, folgt

$$h_{t-i-1} \circ m_{t-i} \neq h'_{t'-i-1} \circ m'_{t'-i}$$

und somit

$$f(h_{t-i-1} \circ m_{t-i}) = h_{t-i} = h'_{t'-i} = f(h'_{t'-i-1} \circ m'_{t'-i}).$$

Also ist wieder eine Kollision von f gefunden.

1.3.4 Digitale Signatur

Digitale Signaturen sind das Pendant zur handschriftlichen Unterschrift und liefern die Authentizität einer Nachricht. Allerdings ist die Nachrichtenlänge eine wesentliche Einschränkung der digitalen Signatur. Diese kann beispielsweise bei RSA nicht länger als der Modul sein (in der Praxis zwischen 1024 und 3072 Bit [7, S.336]). Also werden Hashfunktionen angewendet um dieses Problem zu lösen.

Möchte Bob Alice eine Nachricht schicken, so berechnet er zunächst den Hashwert der Nachricht m und signiert den Hashwert z mit seinem privaten Schlüssel $k_{pr,B}$. Alice berechnet dann den Hashert z' der empfangenen Nachricht m und verifiziert die Signatur s mit Bobs öffentlichem Schlüssel $k_{pub,B}$ (s. Abbildung 3). Hier ist wichtig zu erwähnen, dass die Erzeugung der Signatur und die Verifikation jeweils den Hashwert z als Eingang hat und nicht die eigentliche Nachricht m. Also repräsentiert der Hashwert die Nachricht und wird deswegen auch oft Fingerabdruck der Nachricht genannt.

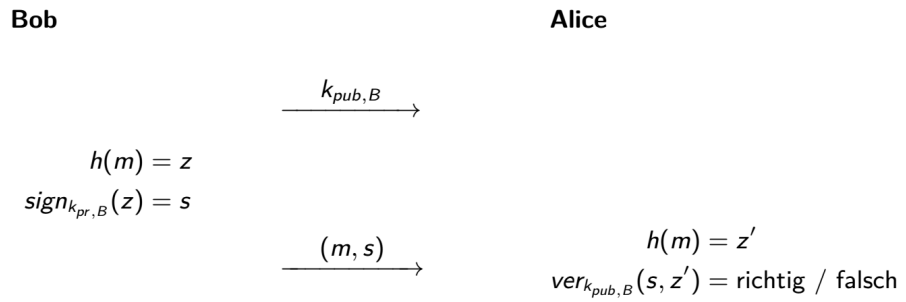

Abbildung 2: Digitale Signaturen mit Hashfunktionen

2 Blockchain

2.1 Blockchain Technologie

Am 31. Oktober 2008 veröffentlichte Satoshi Nakamoto das White Paper "Bitcoin: A Peer – to – Peer Electronic Cash System", die Beschreibung eines kryptografisch abgesicherten Zahlungssystems. Es bietet die Möglichkeit elektronische Zahlungsprozesse ohne eine dritte Partei (Finanzeinrichtungen) zu verarbeiten und basiert auf der Blockchain Technologie. Diese Arbeit befasst sich mit der ältesten Blockchain, der Bitcoin Blockchain.
Die *Blockchain* ist eine Kette aus Blöcken, in denen alle Transaktionen gesammelt werden, also eine Art Buchhaltungssystem. Sie muss von den beteiligten Computern, den Knoten

des Netzwerks, herunter geladen werden. Das heißt es sind Kopien der Transaktionsliste dezentral auf vielen unbekannten Servern gespeichert. Überweisungen werden ohne Intermediär alleine über die Software gesteuert (Peer – to – Peer). Für Überweisungen werden *Kryptowährungen* (z.B. Bitcoin) genutzt. Sie sind das Pendant zu physischem Fiatgeld (Euro, Dollar, Yen) und existieren ausnahmslos in digitaler und verschlüsselter Form. Desweiteren existieren *Wallets*, die den Teilnehmer mit dem dezentralen Netzwerk verbinden. Mit ihrer Hilfe kann man Transaktionen tätigen, den Kontostand überprüfen und teilweise bieten sie die Möglichkeit Währungen miteinander zu tauschen. Das Netzwerk wird mittels Rechenleistung von *Minern* aufrechterhalten.

Der grobe Ablauf eines Transaktionsprozesses wird nun anhand eines Beispiels erklärt. Angenommen Alice möchte Bob Bitcoin überweisen. Zuerst wird ein privater Schlüssel generiert, eine 256 Ziffern lange, zufällig gewählte Binärzahl. Aus diesem öffentlichen Schlüssel wird mittels Elliptischer Kurven der öffentliche Schlüssel generiert. Durch doppeltes Anwenden einer Hashfunktion wird aus diesem öffentlichen Schlüssel schließlich die Bitcoin Adresse erstellt. Also besitzt Alice eine Bitcoin Adresse, mit der sie Bitcoin empfangen oder senden kann. Sie ist öffentlich im Netzwerk als 64 lange Hexadezimalzahl zu sehen. Die Schlüssel werden sicher in den Wallets aufbewahrt. Dann erstellt Alice eine signierte Nachricht, die sie im Netzwerk veröffentlicht. Die Nachricht enthält den Betrag der transferiert werden soll und Bob's Bitcoin – Adresse. Die Miner haben die Aufgabe die Transaktion zu überprüfen und zu bestätigen. Zur Bestätigung der Transaktion müssen sie in einem Konkurrenzkampf mittels Rechenleistung einen Schlüsselcode finden. Wer ihn zuerst findet erhält eine Belohnung in Form von Bitcoin. Der Miner trägt die Transaktion in seine Liste ein und die anderen Listen werden aktualisiert, das heißt die Transaktion wurde bestätigt und ausgeführt. Diese Transaktion ist nun mit einem Zeitstempel im Netzwerk gespeichert und für jeden Teilnehmer einzusehen. Dieser Arbeitsnachweis (Proof of Work) dient dazu, einen vertrauenswürdigen und dezentralen Konsens zu ermöglichen.

2.2 Hashfunktionen in der Blockchain

Für jede einzelne Transaktion, die in dem Netzwerk auftaucht, wird ein Hash berechnet und dann, wie in Abbildung 3 dargestellt, pyramidenförmig weitergerechnet, bis nurnoch die "Wurzel"des Baumes übrig bleibt. Dieser Hashbaum wird dann in einen Block versetzt. Im Durchschnitt wird alle 10 Minuten ein Block generiert, das heißt ein Block enthält alle in den letzten 10 Minuten getätigten Transaktionen. In der Kopfzeile des Blocks befindet sich die Hashwurzel, ein Zeitstempel, der Hash des vorherigen Blocks und die Nonce. Die Nonce zählt die im Hashbaum berechneten Hashes. Nach jedem Hash wird die Nonce um 1

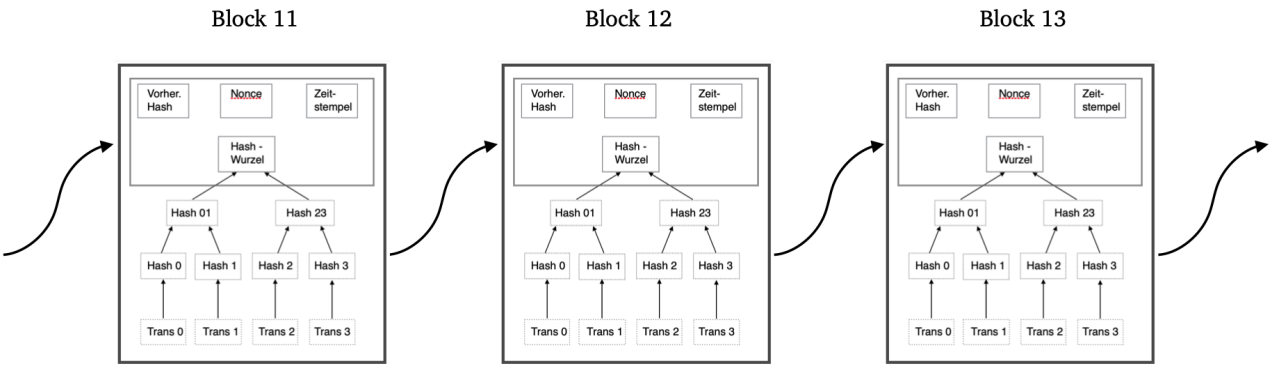

Abbildung 3: Blockkette

erhöht, womit ein neuer Kopfzeilen – Hash berechnet wird. Diese Zahl ist ausschlaggebend für die Gewinnchancen der Miner bei der Schlüsselsuche. Damit die Reihenfolge der Blöcke nicht manipuliert werden kann, wird in die Kopfzeile des Blocks der Hash des vorherigen Blocks geschrieben und somit entsteht eine Blockkette. Bitcoin nutzt beispielsweise den SHA256 – Algorithmus, um Überweisungen in Hash – Blöcken zusammenzufassen. Hashfunktionen bilden gewissermaßen das Rückgrat des Proof of Work Prozesses. Aufgrund der Bestätigung und Erstellung von Hashtransaktionen ist die Blockchain unveränderlich und vor Manipulation geschützt und gilt als bestätigt sicher.

3 Fazit

Die Blockchain Technologie gilt als eine potenzielle neue Basistechnologie der Digitalisierung. Sie löst das Problem der byzantinischen Generäle indem sie das Vertrauen durch Algorithmen und Rechnerleistung ersetzt. Diese Algorithmen sind nur so gut wie die ihr zugrundeliegenden Hashfunktionen. Und damit gehören Hashfunktionen zu den nützlichsten mathematischen Techniken, die es ermöglichen, die Echtheit digitaler Inhalte mit mathematischer Sicherheit nachzuweisen.

Literatur

[1] BEUTELSPACHER, A., NEUMANN, H. B., SCHWARZPAUL, T., 'Kryptografie in Theorie und Praxis: Mathematische Grundlagen für Internetsicherheit, Mobilfunk und elektronisches Geld – 2.Auflage', *Vieweg+Teubner Wiesbaden* (2010).

[2] BITTEL, OLIVER 'Algorithmen und Datenstrukturen AIN 3 – Prof. Dr. Oliver Bittel, HTWG Konstanz' (http:www-home.htwg-konstanz.de/∼bittel/ain_alda/Vorlesung/02_Suchen_Hashverfahren.pdf), Konstanz (2021). [letzter Zugriff am 25. Juni 2021]

[3] BUCHMANN, JOHANNES, 'Einführung in die Kryptologie - 6. Auflage', *Springer Spektrum* (2016).

[4] BSI – BUNDESAMT FÜR SICHERHEIT IN DER INFORMATIONSTECHNIK, 'Kryptographische Verfahren: Empfehlungen und Schlüssellängen', (https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/ TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile) (Stand: 24. März 2021).

[5] KÜSTERS, RALF, WILKE, THOMAS, 'Moderne Kryptographie – Eine Einführung', *Vieweg+Teubner* (2011).

[6] KNEBL, HELMUT, 'Alorithmen und Datenstrukturen – Grundlagen und probabilistische Methoden für den Entwurf und die Analyse – 2. Auflage', *Springer Vieweg* (2021).

[7] PAAR, CHRISTOPH,PELZL, JAN, 'Kryptographie verständlich – Ein Lehrbuch für Studierende und Anwender' *Springer Verlag* (2016).

[8] WITT, KURT-ULRICH, MÜLLER, MARTIN ERIC 'Algorithmische Informationstheorie – Berechenbarkeit und Komplexität verstehen', *Springer Spektrum* (2020).

Abbildungsverzeichnis

4 Anhang

Zeichen	Dezimal	Hexadezimal	Binärsystem
0	48	30	0011 0000
1	49	31	0011 0001
2	50	32	0011 0010
3	51	33	0011 0011
4	52	34	0011 0100
5	53	35	0011 0101
6	54	36	0011 0110
7	55	37	0011 0111
8	56	38	0011 1000
9	57	39	0011 1001
:	58	3A	0011 1010
;	59	3B	0011 1011
<	60	3C	0011 1100
=	61	3D	0011 1101
>	62	3E	0011 1110
?	63	3F	0011 1111
@	64	40	0100 0000
A	65	41	0100 0001
B	66	42	0100 0010
C	67	43	0100 0011
D	68	44	0100 0100
E	69	45	0100 0101
F	70	46	0100 0110
G	71	47	0100 0111
H	72	48	0100 1000
I	73	49	0100 1001
J	74	4A	0100 1010
K	75	4B	0100 1011
L	76	4C	0100 1100
M	77	4D	0100 1101
N	78	4E	0100 1110
O	79	4F	0100 1111
P	80	50	0101 0000
Q	81	51	0101 0001
R	82	52	0101 0010
S	83	53	0101 0011
T	84	54	0101 0100
U	85	55	0101 0101
V	86	56	0101 0110

Zeichen	Dezimal	Hexadezimal	Binärsystem
W	87	57	0101 0111
X	88	58	0101 1000
Y	89	59	0101 1001
Z	90	5A	0101 1010
[	91	5B	0101 1011
\	92	5C	0101 1100
]	93	5D	0101 1101
^	94	5E	0101 1110
_	95	5F	0101 1111
`	96	60	0110 0000
a	97	61	0110 0001
b	98	62	0110 0010
c	99	63	0110 0011
d	100	64	0110 0100
e	101	65	0110 0101
f	102	66	0110 0110
g	103	67	0110 0111
h	104	68	0110 1000
i	105	69	0110 1001
j	106	6A	0110 1010
k	107	6B	0110 1011
l	108	6C	0110 1100
m	109	6D	0110 1101
n	110	6E	0110 1110
o	111	6F	0110 1111
p	112	70	0111 0000
q	113	71	0111 0001
r	114	72	0111 0010
s	115	73	0111 0011
t	116	74	0111 0100
u	117	75	0111 0101
v	118	76	0111 0110
w	119	77	0111 0111
x	120	78	0111 1000
y	121	79	0111 1001
z	122	7A	0111 1010
{	123	7B	0111 1011
|	124	7C	0111 1100
}	125	7D	0111 1101

Abbildung 4: ASCII Tabelle