Blockchain Engineering

Hash-Funktionen

Dr. Lars Brünjes

GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung

MODULARES INNOVATIVES
NETZWERK FÜR DURCHLÄSSIGKEIT

AUFSTIEG DURCH
BILDUNG >>
OFFENE HOCHSCHULEN

26. September 2019

Eine Hash-Funktion ist eine Funktion H, die Bytestrings beliebiger Länge auf Bytestrings mit fester Länge (sogenannte Digests) abbildet.
Übliche Digest-Längen sind z.B. 128, 256 oder 512 Bits bzw. 16, 32 oder 64 Bytes.

Anwendungen von Hashing

- Prüfsummen
- Commitments (Verpflichtungen)
- Blockchain
- Merkle-Trees (-Bäume)

- Prüfsummen
  - Hat sich eine Datei verändert?
  - Habe ich eine Datei fehlerfrei kopiert/heruntergeladen?
- Commitments (Verpflichtungen)
- Blockchain
- Merkle-Trees (-Bäume)

- Prüfsummen
- Commitments (Verpflichtungen)
  - Öffentliches Festlegen auf eine Wahl.
  - Kein Offenlegen dieser Wahl.
  - Unmöglich, die Wahl später zu ändern.
- Blockchain
- Merkle-Trees (-Bäume)

- Prüfsummen
- Commitments (Verpflichtungen)
  - Öffentliches Festlegen auf eine Wahl.
  - Kein Offenlegen dieser Wahl.
  - Unmöglich, die Wahl später zu ändern.
- Blockchain
- Merkle-Trees (-Bäume)

Wer ist der Mörder?

Sie und ein Freund schauen einen Krimi. Sie glauben zu wissen, wer der Mörder ist und wollen vor Ihrem Freund damit angeben, ihm aber nicht den Spaß verderben. Sie können “Der Gärtner ist der Mörder” auf einen Zettel schreiben und Ihrem Freund zustecken. Nach dem Film öffnet er den Zettel und sieht, dass Sie den Mörder vorher kannten.

- Prüfsummen
- Commitments (Verpflichtungen)
  - Öffentliches Festlegen auf eine Wahl.
  - Kein Offenlegen dieser Wahl.
  - Unmöglich, die Wahl später zu ändern.
- Blockchain
- Merkle-Trees (-Bäume)

Wer ist der Mörder — auf Entfernung?

Sie und Ihr Freund schauen wieder einen Krimi, sind aber räumlich getrennt. Sie wissen wieder, wer der Mörder ist. Sie können “Der Koch ist der Mörder” *hashen* und Ihrem Freund den Hash schicken. Wenn Sie später den Originaltext schicken, beweist das, dass Sie den Mörder vorher kannten.

- Prüfsummen
- Commitments (Verpflichtungen)
- Blockchain
  - Jeder Block enthält den Hash des (Kopfes/Headers) des *vorigen* Blocks in der Kette.
  - In Bitcoin und anderen Krypto-Währungen enthält jeder Block die sogenannte Merkle Wurzel, den Hash der Wurzel des Merkle-Baumes, der alle Transaktionen des Blocks enthält.
- Merkle-Trees (-Bäume)

- Prüfsummen
- Commitments (Verpflichtungen)
- Blockchain
- Merkle-Trees (-Bäume)
  - Beispiel einer sogenannten authentifizierten Datenstruktur: Klienten können Fragen an den Besitzer der Datenstruktur stellen und sich auf die Antworten verlassen, ohne die Daten selbst halten zu müssen.
  - In Bitcoin haben nur “Volle”-Knoten (full nodes) den gesamten Baum aller Transaktionen gespeichert, während SPV-Knoten (Simple Payment Verification) nur die Merkle-Wurzel kennen. Trotzdem können volle Knoten SPV-Knoten mit Hilfe eines sogenannten Merkle-Pfades beweisen, dass bestimmte Transaktionen im Baum gespeichert sind.

- Prüfsummen
- Commitments (Verpflichtungen)
- Blockchain
- Merkle-Trees (-Bäume)

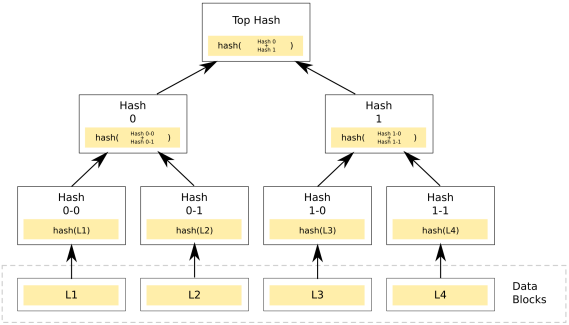

Abb.: Merkle-Baum

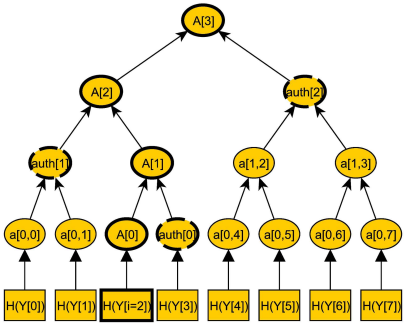

Abb.: Merkle-Pfad

- “Message Digest” Nr. 5.
- Entworfen 1992 von *Ronald Rivest* (dem “R” in “RSA”).
- Größe: 128 Bit
- Nicht sicher! Kollisionen können in wenigen Sekunden, Kollisionen mit vorgegebenem Prefix in wenigen Stunden gefunden werden!
- Trotzdem immer noch (2019) weit verbreitet...

- Online: `https://emn178.github.io/online-tools/md5.html`
- In Python:

```
"""
>>> compute_hash(b'Blockchain Technologie ist toll!', 'MD5').hex()
'bbce1b298a68819f1a86e052a58f7137'


>>> compute_hash(b'Blockchain Technologie ist toll.', 'MD5').hex()
'92acb5cdb3f399df0c26673c7e140ceb'
"""

from rsa import compute_hash
```

- Im Linux Terminal:

```
echo -n "Blockchain Technologie ist toll!" | md5sum
```

- Veröffentlicht 2001 von der *National Security Agency (NSA)*.
- Größe: 256 Bit
- Wird von Bitcoin und Cardano benutzt.
- Gilt (noch) als sicher.

- Online: `https://emn178.github.io/online-tools/sha256.html`
- In Python:

```
"""
>>> compute_hash(b'Blockchain Technologie ist toll!', 'SHA-256').hex()
'a72706a2cd9e0c3b50336ff83e1fd0513fddbe36877fb10c944bbd84f8565973'

>>> compute_hash(b'Blockchain Technologie ist toll.', 'SHA-256').hex()
'a9335ee9ccd886a93fccb2b47909cc2a7dd9b17f8a7990149cd05d7d2c065407'
"""

from rsa import compute_hash
```

- Im Linux Terminal:

```
echo -n "Blockchain Technologie ist toll!" | sha256sum
```

Eine Hash-Funktion H ist kryptografisch sicher (cryptographically secure), wenn sie die folgenden Eigenschaften hat:

- effektiv berechenbar,
- kollisionsfrei,
- hiding (versteckend)
- Puzzle-freundlich.

Eine Hash-Funktion H ist effektiv berechenbar wenn ihre Berechnung schnell ist (auch für große Input-Strings).
Natürlich wird die Rechenzeit mit der Länge des Input-Strings steigen, aber sie sollte linear (oder höchstens polynomial) steigen, nicht exponentiell.

Eine Hash-Funktion H ist kollisionsfrei, wenn es praktisch unmöglich ist, zu gegebenem h aus dem Bild von H Bytestrings x und y mit $x \neq y$ und $H(x) = H(y)$ zu finden.

Injektivität

Da H *beliebig lange* Bytestrings als Eingabe akzeptiert und nur eine endliche (wenn auch eventuell sehr große) Anzahl von möglichen Ausgaben hat, kann H nicht injektiv sein, und es muss unendlich viele Kollisionen geben. Kollisionsfreiheit bedeutet *nicht*, dass es keine Kollisionen gibt, sondern nur, dass Kollisionen sehr schwer zu finden sind.

Eine Hash-Funktion H ist hiding, wenn es für Bytestrings r und x praktisch unmöglich ist, x aus $H(r \mid x)$ zu rekonstruieren, sofern r aus einer “hinreichend zufälligen” Verteilung zufällig gewählt wurde.

Schere, Stein, Papier

Ich will H benutzen, um remote “Schere, Stein, Papier” zu spielen. Ich treffe meine Wahl, z.B. “Stein”, schicke dann meinem Partner $H('Stein')$. Da es nur drei Möglichkeiten gibt, könnte der einfach meinen Hash mit $H('Schere')$, $H('Stein')$ und $H('Papier')$ vergleichen, um zu wissen, was ich spiele. Da H aber hiding ist, kann ich das verhindern, wenn ich meine Wahl mit einer langen Zufallszahl verkette und zum Beispiel $H('375823748237Stein')$ schicke.

Eine Hash-Funktion H ist Puzzle-freundlich, wenn es sehr aufwendig ist, zu gegebenem r (zufällig aus einer “hinreichend zufälligen” Verteilung gezogen) ein x zu finden, so dass $H(r \mid x)$ eine gegebene Eigenschaft hat.

Bitcoin Mining

In Bitcoin versuchen Miner, bei vorgegebenem r (dem Block-Header) ein x (die Nonce) zu finden, so dass $H(r \mid x)$ mit einer gewissen Mindestanzahl von Nullen beginnt.
Wenn H Puzzle-freundlich ist, gibt es für Miner keine bessere Strategie, als all möglichen x der Reihe nach auszuprobieren.

Hinweis

Diese Publikation wurde im Rahmen des vom Bundesministerium für Bildung und Forschung (BMBF) geförderten Bund- Länder- Wettbewerbs “Aufstieg durch Bildung: offene Hochschulen” erstellt. Die in dieser Publikation dargelegten Ergebnisse und Interpretationen liegen in der alleinigen Verantwortung der Autor/innen.