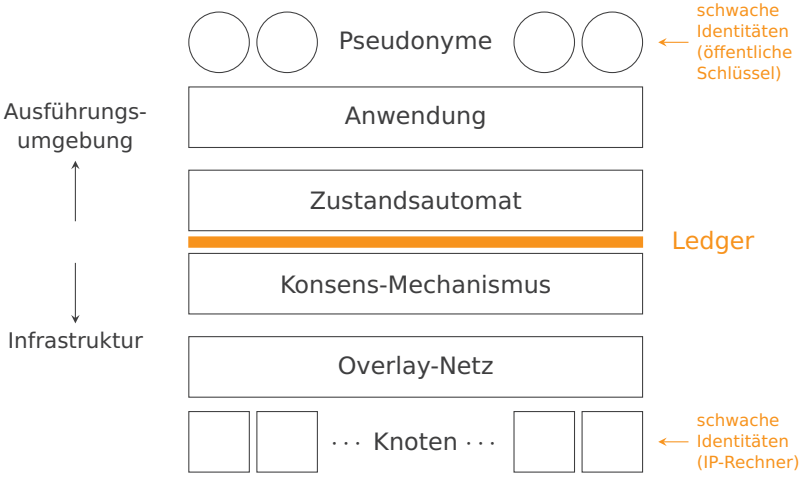
universität innsbruck

Prinzipien von Blockchain-Systemen

Transaktionslogik in Bitcoin und Ethereum

Rainer Böhme

Schichtmodell für Blockchain-Systeme



Einheiten

Die Transaktionslogik regelt u. a. die Übertragung von virtuellen Gütern.

Alle bekannten Blockchain-Systeme definieren Währungseinheiten zur Realisierung eines Anreizsystems. Ein rechtlicher Status als Zahlungsmittel ergibt sich daraus nicht.

Definition				**Marktpreise**	
	1	Bitcoin	$\approx$	7 000	€
=	10^8	Satoshi	$\approx$	0.007	Cent
	1	Ether	$\approx$	175	€
=	10^3	Finney			
=	10^6	Szabo	$\approx$	0.018	Cent
=	10^{18}	Wei			

Stand: 24.04.2020

Auswahl der Blockchain-Systeme

#	Name	Market Cap	Price	Volume (24h)	Circulating Supply	Change (24h)	Price Graph (7d)
1	Bitcoin	€130.914.351.104	€7.135,16	€34.984.359.661	18.347.775 BTC	1,70%	
2	Ethereum	€20.135.335.380	€181,91	€18.437.188.829	110.689.239 ETH	0,76%	
3	XRP	€8.031.795.183	€0,182170	€1.732.690.531	44.089.620.959 XRP *	1,07%	
4	Tether	€5.903.517.153	€0,928075	€48.615.194.855	6.361.032.509 USDT *	-0,11%	
5	Bitcoin Cash	€4.136.613.105	€224,92	€3.728.469.640	18.391.325 BCH	1,16%	

Bitcoin und Ethereum sind Repräsentanten unterschiedlicher Entwurfsphilosophien.

Quelle: coinmarketcap.com, 27.04.2020

Blockchain-Knowhow ist relativ

Beispiel: Begriff der „Adresse“

Adressen bei Bitcoin

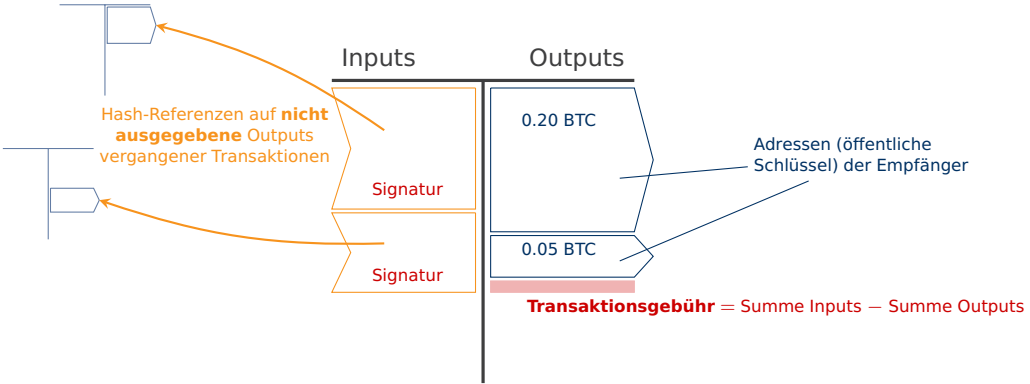
- Commitment zu einem öffentlichen Schlüssel (P2PKH) oder Skript (P2SH)
- 25 Byte: $(0 \| \text{RIPEMD-160}(\text{SHA-256}(\boldsymbol{t})) \| \text{Prüfsumme})$ in Base58-Kodierung
- z. B. `183hmJGRuTEi2YDCWy5iozY8rZtFwVgahM`

Adressen bei Ethereum

- 20 Byte: $(\text{Keccak}(\boldsymbol{x}) \,\&\, (2^{161} - 1))$ in Hex-Darstellung
- Wobei $\boldsymbol{x} \triangleq \boldsymbol{t}$ für EOAs und $\boldsymbol{x} \triangleq (\text{Sender} \| \text{nonce++})$ für CAs.
- z. B. `0xBB9bc244D798123fDe783fCc1C72d3Bb8C189413`

Transaktionskodierung bei Bitcoin

Jede Bitcoin-Transaktion besteht aus einer Liste aus Inputs und einer Liste aus Outputs.



Optional: `nLockTime`-Feld gibt Zeitpunkt in Blockhöhe oder Realzeit an, vor der die Transaktion ungültig ist.

Einfache Formalisierung

Wie verwenden die Notation

$$T_n = \Big(\overbrace{\underbrace{(T_{i,3}, T_{j,1})}_{\text{Input-Liste}}, \underbrace{((0.2, \boldsymbol{t}_{\text{Alice}}), (0.05, \boldsymbol{t}_{\text{Bob}}))}_{\text{Output-Liste}}}^{T'_n}, \underbrace{(\boldsymbol{s}_{\text{Bob}}(T'_n), \boldsymbol{s}_{\text{Bob}}(T'_n))}_{\text{Signaturen}} \Big)$$

mit folgenden Konventionen:

- $T_{i,3}$ ist eine Referenz auf den 3. Output von T_i.
- Der erste Wert jedes Output-Tupels ist der Betrag.
- $\boldsymbol{s}_{\text{Bob}}$ und $\boldsymbol{t}_{\text{Bob}}$ sind Signier- bzw. Testschlüssel unter der Kontrolle von Bob.
- Wenn Bob mehrere Schlüsselpaare hat, schreiben wir z. B.: $\boldsymbol{t}^1_{\text{Bob}}$, $\boldsymbol{t}^2_{\text{Bob}}$, ...

T' kann auf mehrere Arten gebildet werden: `https://bitcoin.org/en/developer-guide#signature-hash-types`

Beispiel

Bob besitzt 1.2 BTC aus dem 1. Output von T_1
0.3 BTC aus dem Wechselgeld (2. Output) von T_2
0.5 BTC aus dem Wechselgeld (4. Output) von T_3

Bob nutzt $\boldsymbol{t}^2_{\text{Bob}}$ als Wechselgeldadresse. Er möchte Alice 1.5 BTC an $\boldsymbol{t}_{\text{Alice}}$ überweisen:

$$T_4 = \big((T_{1,1}, T_{3,4}), \big((1.5, \boldsymbol{t}_{\text{Alice}}), (0.15, \boldsymbol{t}^2_{\text{Bob}})\big), (\boldsymbol{s}_{\text{Bob}}(T'_4), \boldsymbol{s}^2_{\text{Bob}}(T'_4))\big)$$

Alice verschiebt 1.0 BTC zur Sicherheit in ihre eigene *paper wallet* mit Schlüssel $\boldsymbol{t}^2_{\text{Alice}}$:

$$T_5 = \big((T_{4,1}), \big((1.0, \boldsymbol{t}^2_{\text{Alice}}), (0.45, \boldsymbol{t}_{\text{Alice}})\big), (\boldsymbol{s}_{\text{Alice}}(T'_5))\big)$$

Bob möchte sein Wechselgeld konsolidieren. Ist folgende Transaktion gültig?

$$T_6 = \big((T_{2,2}, T_{3,4}), ((0.4, \boldsymbol{t}_{\text{Bob}})), (\boldsymbol{s}^2_{\text{Bob}}(T'_6), \boldsymbol{s}^2_{\text{Bob}}(T'_6))\big)$$

Coinbase-Transaktionen

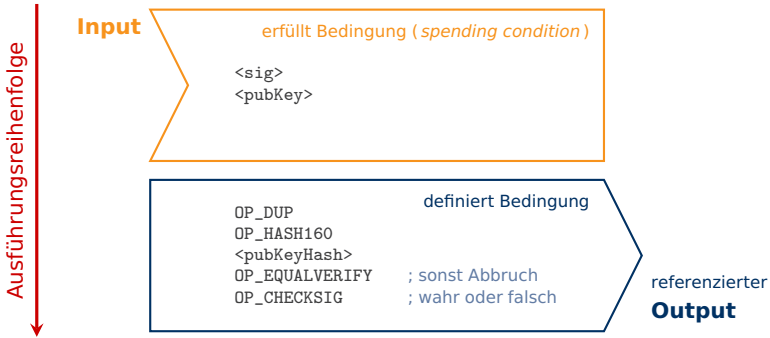
$$T_0 = \big((50 + \text{Gebühren}), ((50 + \text{Gebühren}, \boldsymbol{t}_{\text{Pool-Manager}})), ()\big)$$

Spezialfall

- Die erste Transaktion jedes Blocks weist die Belohnung und die Summe aller im Block enthaltenen Transaktionsgebühren dem Leader zu.
- Diese Coinbase-Transaktionen haben keine anderen Inputs und keine Signaturen.
- Outputs von Coinbase-Transaktionen dürfen erst nach 100 Blöcken ausgegeben werden (*cooldown*).

Bitcoin-Transaktionsskripte

- Bitcoin implementiert eine Stapel-basierte virtuelle Maschine, die Forth-ähnliche Programme evaluiert um über die Verwendung von Outputs zu entscheiden.
- Die Sprache ist mit Absicht nicht Turing-vollständig.



Den vollständigen Befehlssatz finden Sie hier: `https://en.bitcoin.it/wiki/Script`

Kleine Typologie der Output-Skripte

Pay-to-PubKeyHash (P2PKH)

- Konvention für Standard-Output-Skripte: Commitment zu öffentlichem Schlüssel
- Client-Unterstützung durch Adressen mit dem Prefix `00` $\rightarrow$ `1` in Base58-Kodierung

Pay-to-ScriptHash (P2SH)

- Commitment zu Input-Skript
- Client-Unterstützung durch Adressen mit dem Prefix `05` $\rightarrow$ `3` in Base58-Kodierung
- Signaturen sind trotzdem dringend empfohlen. Sie können aber flexibler gestaltet sein, z. B. *n* gültige Signaturen zu *m* Testschlüsseln verlangen (`OP_CHECKMULTISIG`).
- Zeitabhängige Bedingungen ermöglichen, einzelne Outputs nach einem definierten Zeitpunkt (anders) ausgeben zu können (`OP_CHECKLOCKTIMEVERIFY`).

Generisches Datenfeld

- Die `OP_RETURN`-Instruktion „verbrennt“ den Output und speichert dafür bis zu 80 Byte Nutzdaten in die Blockchain.

UTXO-Modell versus Kontenmodell

Jargon: TX = Transaction, TXO = Transaction Output, UTXO = Unspent TXO

UTXO-Modell

- „Rechnet in Transaktionen“, Münzanalogie
- Benötigt Wechselgeld
- Outputs gliedern Transaktionen in kleinere, fest definierte Einheiten.
- Client-Datenhaltung optimiert zur Bestimmung der Neuheit von Outputs, jedoch keine Salden
- Wert ergibt sich aus Rückverfolgbarkeit jeder Einheit zur Coinbase-Transaktion.
- Nicht fungibel

bei Bitcoin im Einsatz

Kontenmodell

- „Rechnet in Konten“, Giralgeldanalogie
- Kein Wechselgeld nötig
- Transaktionen ähneln Nachrichten mit Empfängern und Parametern.
- Client schreibt Systemzustand fort und prüft Salden.
- Zähler zur Bestimmung der Neuheit.
- Wert ergibt sich aus Korrektheit aller Zustandsübergänge.
- Fungibilität möglich

bei Ethereum im Einsatz

Ethereum Virtual Machine (EVM)

Architektur
Jeder Ethereum-Knoten implementiert eine Turing-vollständige virtuelle Maschine mit

- stapelbasierter Architektur und
- Wortbreite 32 Byte (256 Bit).

Einbindung in die Transaktionslogik
Im Gegensatz zum UTXO-Modell von Bitcoin kann die EVM

- Zustandsübergänge im Rahmen der Nebenbedingungen frei definieren und
- den Zustandsraum erweitern.

Programmierung
Für die EVM existieren Hochsprachen: Solidity, LLL, Serpent (nicht empfohlen)

Warum wollen „Normalnutzer“ für die EVM programmieren [können]?

Motivation von „Smart Contracts“

Kontext

- Es gibt viele Versuche, Rechtsquellen (Gesetze, Verwaltungsakte, Verträge, . . .) zu formalisieren um sie dann maschinengestützt zu analysieren.
- **Probleme:** natürliche Sprache, bewusste Mehrdeutigkeit, Interpretationsspielraum

Der umgekehrte Weg

Wenn Parteien

- Vertragstexte direkt in einer **formalen Sprache** schreiben und
- eine **vertrauenswürdige Ausführungsumgebung** zur Verfügung steht,

dann kann diese die Einhaltung der Vertragsbedingungen automatisch überprüfen.

Sog. „Smart Contracts“ sind Programme, die korrespondierende Verträge **innerhalb des Zustandsraums** eines Blockchain-Systems **durchsetzen** können.

Szabo, N.: Formalizing and securing relationships on public networks. *First Monday* 2 (9), 1997.

Beispiel

Alice und Bob schließen einen Vertrag:

> *„Wir werfen die Münze. Der Gewinner bekommt vom Verlierer ein Geldstück.“*

Bemerkungen

- Juristisch liegt ein Vertrag vor, sobald zwei Willenserklärungen abgegeben wurden. Ob dies nachweisbar oder durchsetzbar ist, spielt zunächst keine Rolle.
- Wir kennen eine Methode (kryptographische Commitments), um den Münzwurf gerecht und ohne dritte Partei über ein Rechnernetz durchzuführen.
- Das reicht aber nicht aus um sicherzustellen, dass der Verlierer tatsächlich zahlt.
- Dieser Vertrag lässt sich in einem Blockchain-System abbilden, sodass die Zahlung (in Einheiten eines virtuellen Guts) nach dem Wurf unabwendbar ist.

Kontentypen bei Ethereum

Ethereum unterscheidet zwei Typen von Konten (engl. *account*).

Parteien „besitzen“:

Externally Owned Account (EOA)

gesteuert durch Signierschlüssel

- Adresse
- Saldo (*balance*)
- Transaktionszähler (*nonce*)

Transaktionen referenzieren:

Code Account (CA)

gesteuert durch Programmlogik

- Adresse
- Bytecode
- Saldo (*balance*)
- Lokale Variablen (Zustand)
- Kind-CA-Zähler (*nonce*)

„Inter-Konten-Kommunikation“

Ethereum nutzt ein Nachrichtensystem zur Kommunikation zwischen Konten.
Abhängig vom Kontotyp werden die Nachrichten unterschiedlich bezeichnet.

EOAs lösen aus:

Transaktionen

- Empfänger
- Betrag
- Daten
- Gebühreninformation
- Digitale Signatur

in der Blockchain gespeichert

CAs lösen aus:

Message Calls

- Empfänger
- Betrag
- Daten
- Gebühreninformation

deterministische Folge von Transaktionen

Lebenszyklus von Code Accounts

Erstellen

- Die EVM interpretiert die Daten von Transaktionen ohne Empfängerangabe als Bytecode, führt ihn aus und speichert die Ausgabe als neuen Code Account.
- Deployment-Konvention: (Initialisierungs-Code‖Code‖Parameter)

Aufrufen

- Über Transaktionen und Message Calls, Parameter werden im Datenfeld übergeben

Aktualisieren

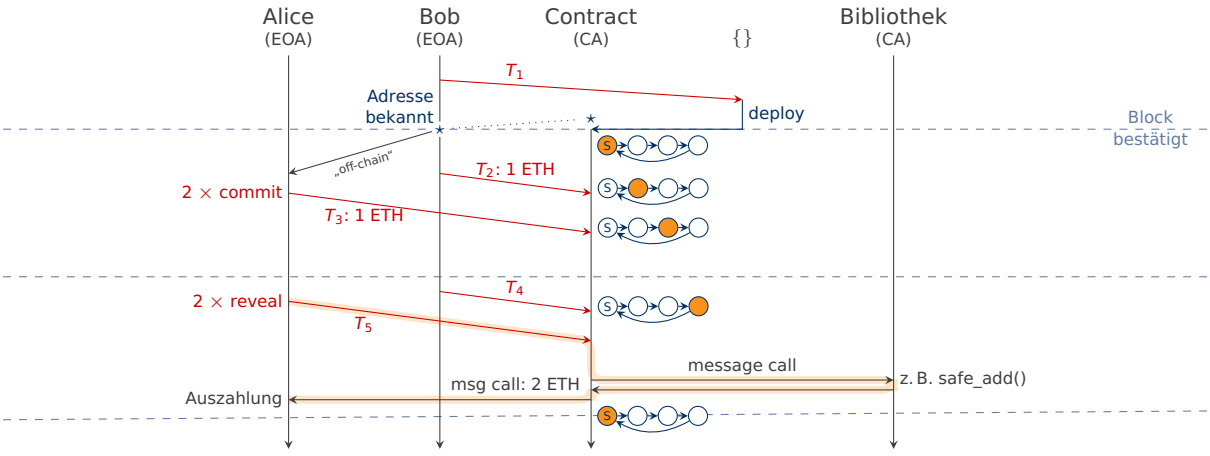
- Nicht vorgesehen → Workaround über Proxy-Pattern, falls erwünscht!

Löschen

- Nur über die EVM-Instruktion `SELFDESTRUCT`
- Danach erscheint die Adresse leer → Aufrufe geben `TRUE` ohne Seiteneffekt zurück!

Beispiel

Alice und Bob möchten durchsetzbar eine Münze werfen.



Transaktionsgebühren bei Ethereum

Ziele:

1. Entschädigung für Ressourcenbereitstellung
2. Beschränkung der Turing-vollständigen Ausführungsumgebung

Jede EVM-Instruktion hat einen Preis in der Recheneinheit „Gas“.
Der Umrechnungskurs von Gas zu Ether ist marktbestimmt.

$$\text{Gebühr} = \text{verbrauchtes Gas} \times \text{Gaspreis}$$

Auszug

0x01 ADD	3
0x06 MOD	5
0x20 KECCAK	30
0xf0 CREATE	40

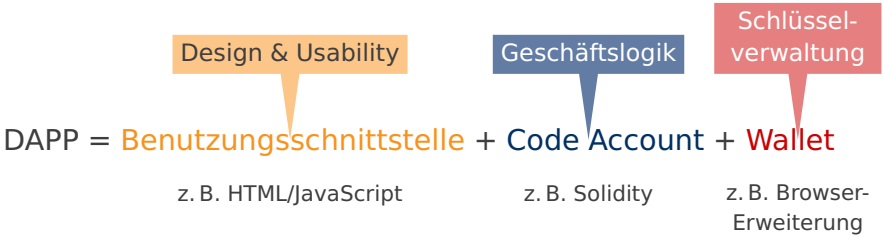
Benötigt eine Berechnung mehr Gas als vom Nutzer zur Verfügung gestellt, dann

- wird die Transaktion abgelehnt (Rollback) und
- der Miner erhält die Gebühr.

→ Neue Zielfunktion für Code-Optimierung: minimale Kosten

Aktuelle Gaspreise und Statistiken: `http://ethgasstation.info`

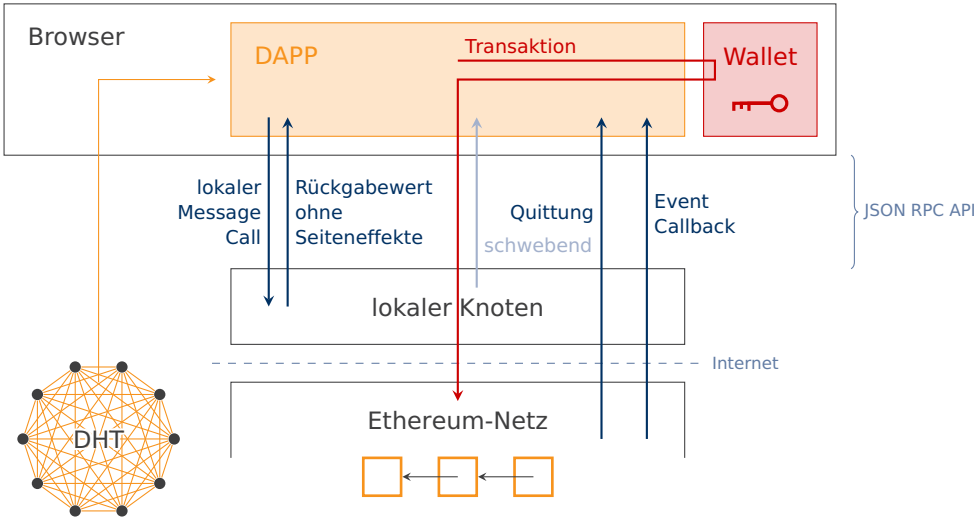
Decentralized Application (DAPP)



Vertrauensanker

- Der Code Account regelt die Sicherheitsinteressen aller.
- Die Sicherheit des Wallets ist notwendig für die eigene Sicherheit.
- Die Benutzungsschnittstelle definiert die eigene Sicht auf die Interaktion. Sicherheitsrelevante Entscheidungen bedürfen korrekter Information.

Architektur für DAPPs auf Ethereum



Wichtige Einrichtungen im Ethereum-Ökosystem

Swarm

- Verteilte Datenbank, die beliebige Informationen verfügbar hält.
- Im Gegensatz zur Blockchain erfolgt keine Serialisierung und Gültigkeitsprüfung.
- Adressierung über Hashwerte
- Anreizsystem und Abrechnung in Ether

Ethereum Name Service (ENS)

- Verteilte Datenbank, die menschenlesbare Namen zu Ethereum-Adressen auflöst.
- Anreizsystem und Abrechnung in Ether

Orakel

- Dienste, die Ereignisse in der Realwelt kodieren und sie Code Accounts als Entscheidungsgrundlage zur Verfügung stellen.
- Schwer dezentralisierbar: Vertrauen in eine (wenige) verantwortliche Partei(en).

Syllabus

05.03.20	1. Einführung und Grundlagen	
23.04.20	2. Infrastruktur für Blockchain-Systeme	
30.04.20	3. Transaktionslogik in Bitcoin und Ethereum	
07.05.20	Übung: Blockchain-Analyse mit BlockSci	(Martin Plattner)
14.05.20	4. Datenschutz und Sicherheit	
28.05.20	Übung: Ethereum-Programmierung mit Solidity	(Michael Fröwis)
04.06.20	5. Skalierbarkeit, Off-Chain-Transaktionen, Governance	
18.06.20	6. Wiederholung, Fragestunde	
25.06.20	Klausur	

Stand: 30. April 2020. Änderungen vorbehalten.