Ethereum: Smart Contracts & Varianten

Lars Hupel
Software Architecture Summit
2019-03-12

INNOQ

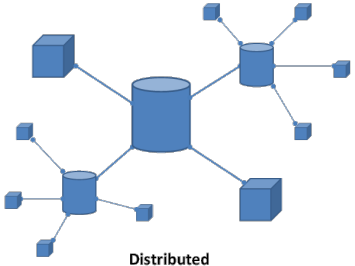
Ethereum

- eine der jüngeren Blockchains
- Währung: Ether (ETH)
- Platz 2–3 in Marktkapitalisierung
- technische Neuerung: Plattform für *Decentralized Apps*

Decentralized Apps

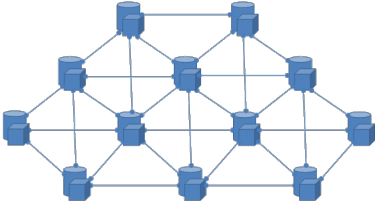
Centralized

one node does everything



Distributed

nodes distribute work to sub-nodes



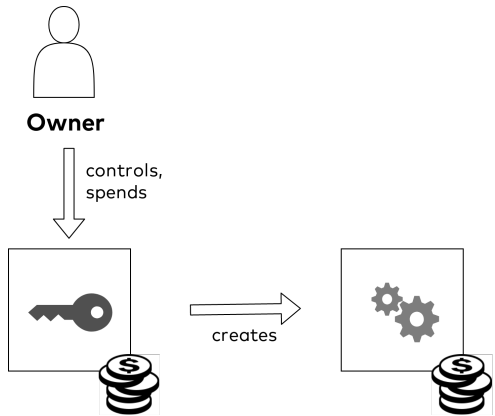
Decentralized

nodes are only connected to peers

Ethereum

Zwei Arten von Accounts:

1. Externally Owned
2. Smart Contracts



Smart Contracts

Smart Contracts

... sind spezielle Accounts mit eigener Identität und Kontostand

Smart Contracts

- ... sind spezielle Accounts mit eigener Identität und Kontostand
- ... haben keine Ownership

Smart Contracts

- ... sind spezielle Accounts mit eigener Identität und Kontostand
- ... haben keine Ownership
- ... enthalten Speicherzellen mit veränderlichen Werten

Smart Contracts

- ... sind spezielle Accounts mit eigener Identität und Kontostand
- ... haben keine Ownership
- ... enthalten Speicherzellen mit veränderlichen Werten
- ... bestehen aus unveränderlichem Code

Smart Contracts

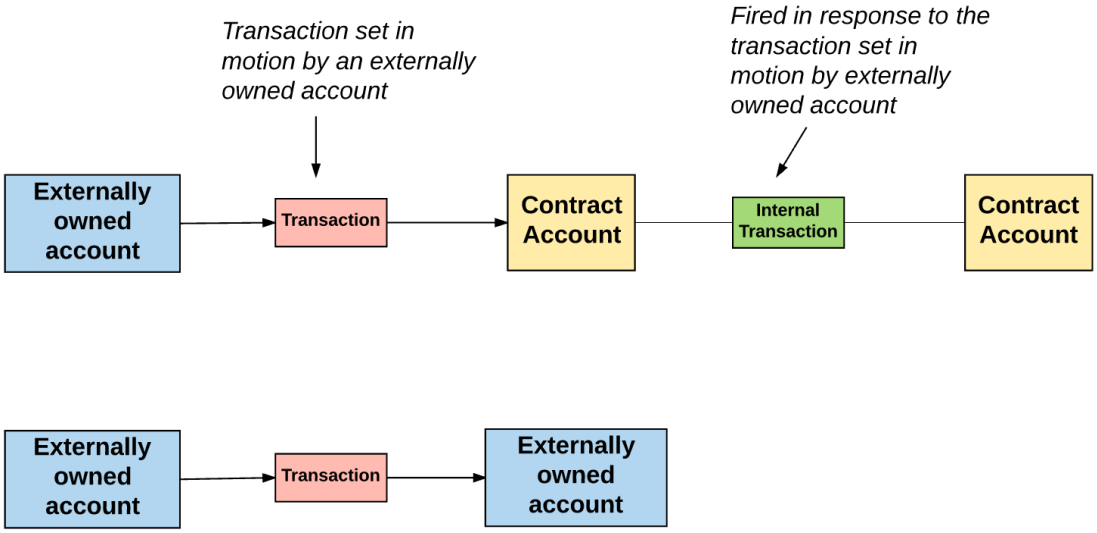
- … sind spezielle Accounts mit eigener Identität und Kontostand
- … haben keine Ownership
- … enthalten Speicherzellen mit veränderlichen Werten
- … bestehen aus unveränderlichem Code
- … können mit anderen Contracts interagieren, werden aber nie selbstständig aktiv

Vergleich: **digitaler Notar**

- Rechte und Pflichten vertraglich festgelegt
- neutrale Instanz überwacht Ablauf
- Interaktion mit dritten Parteien

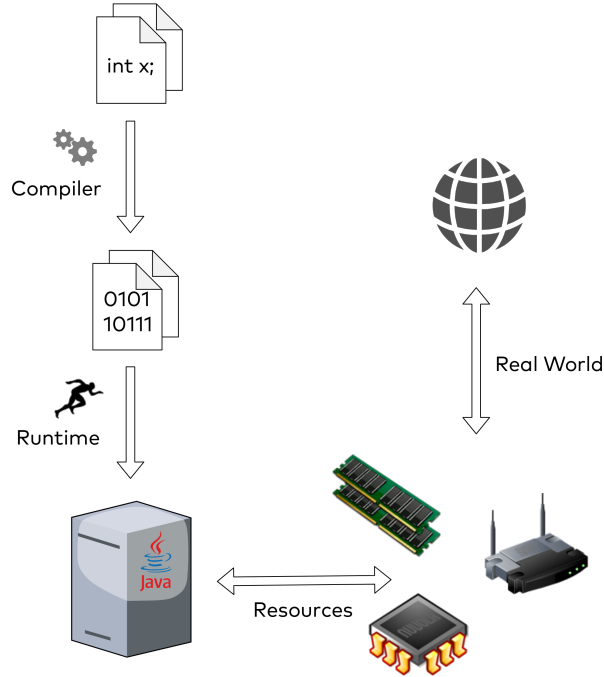
Transaktionsarten

Payment	Übertrag von Geld in einen Contract oder EOA
Call	Aufruf einer Funktion
Internal	Contract interagiert mit anderem Contract


Transaction set in motion by an externally owned account
Fired in response to the transaction set in motion by externally owned account
Externally owned account
Transaction
Contract Account
Internal Transaction
Contract Account
Externally owned account
Transaction
Externally owned account

Ethereum Virtual Machine

Virtuelle Maschinen
sind Mischformen
zwischen Interpreter
und Compiler



Ablauf einer Transaktion

Ablauf einer Transaktion

1. User erzeugt Transaktion:
   - Signatur (enthält Quelle)
   - Ziel
   - Geldbetrag
   - Daten

Ablauf einer Transaktion

1. User erzeugt Transaktion:
   - Signatur (enthält Quelle)
   - Ziel
   - Geldbetrag
   - Daten
2. Miner überprüfen Gültigkeit

Ablauf einer Transaktion

1. User erzeugt Transaktion:
   - Signatur (enthält Quelle)
   - Ziel
   - Geldbetrag
   - Daten
2. Miner überprüfen Gültigkeit
3. Miner führen Code aus, falls Ziel ein Contract ist

Ablauf einer Transaktion

1. User erzeugt Transaktion:
   - Signatur (enthält Quelle)
   - Ziel
   - Geldbetrag
   - Daten
2. Miner überprüfen Gültigkeit
3. Miner führen Code aus, falls Ziel ein Contract ist
4. Persistieren der Änderungen in die Blockchain

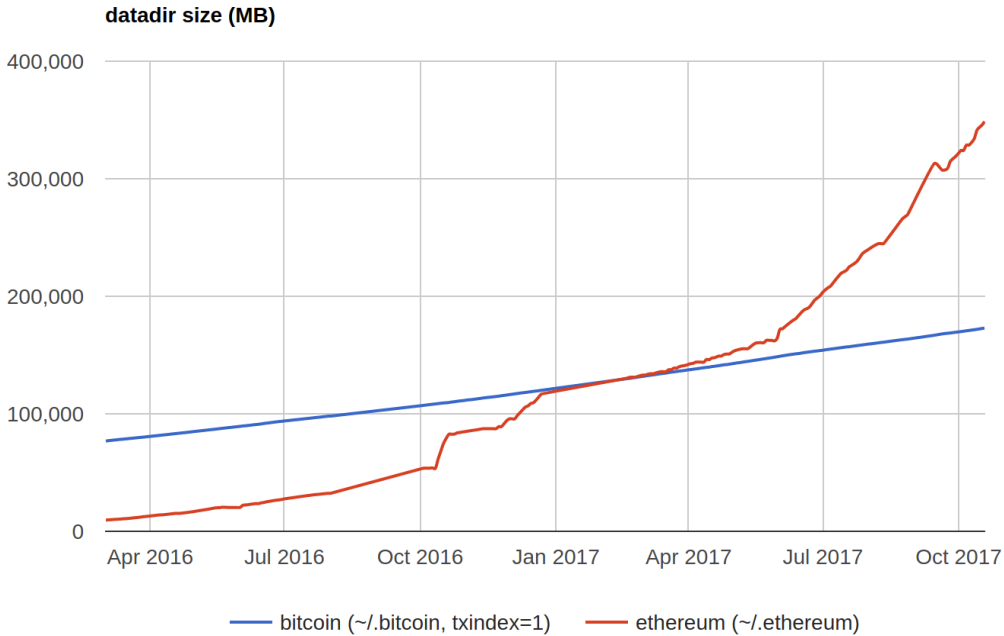
```
interface Transaction {
  // signature: contains public key of EOA
  Signature getSignature();

  // recipient address, can be any address type
  Address getRecipient();

  // amount of Ether to be sent, can be 0
  Value getValue();

  // data for a call, can be empty
  String getData();

  // internal TXNs
  List<Transaction> getInternalTxns();
}
```


datadir size (MB)
400,000
300,000
200,000
100,000
0
Apr 2016
Jul 2016
Oct 2016
Jan 2017
Apr 2017
Jul 2017
Oct 2017
bitcoin (~/.bitcoin, txindex=1)
ethereum (~/.ethereum)

Was kann die EVM?

1. arithmetische Operationen
2. bitweise & logische Operationen
3. Hashes (meist SHA-3)
4. Umgebungsinformation (Kontostand, Aufrufer, ...)
5. Blockinformation (Zeitstempel, Blocknummer, ...)
6. Stack- und Speicheroperationen
7. Sprünge
8. Logging
9. Kontooperationen (Erzeugung, Zahlung, Aufruf, ...)

0s: Stop and Arithmetic Operations

All arithmetic is modulo 2^{256} unless otherwise noted. The zero-th power of zero 0^0 is defined to be one.

Value	Mnemonic	δ	α	Description
0x00	STOP	0	0	Halts execution.
0x01	ADD	2	1	Addition operation. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{\mu}_s[0] + \boldsymbol{\mu}_s[1]$
0x02	MUL	2	1	Multiplication operation. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{\mu}_s[0] \times \boldsymbol{\mu}_s[1]$
0x03	SUB	2	1	Subtraction operation. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{\mu}_s[0] - \boldsymbol{\mu}_s[1]$
0x04	DIV	2	1	Integer division operation. $\boldsymbol{\mu}'_s[0] \equiv \begin{cases} 0 & \text{if} \quad \boldsymbol{\mu}_s[1] = 0 \\ \lfloor \boldsymbol{\mu}_s[0] \div \boldsymbol{\mu}_s[1] \rfloor & \text{otherwise} \end{cases}$
0x05	SDIV	2	1	Signed integer division operation (truncated). $\boldsymbol{\mu}'_s[0] \equiv \begin{cases} 0 & \text{if } \boldsymbol{\mu}_s[1] = 0 \\ -2^{255} & \text{if } \boldsymbol{\mu}_s[0] = -2^{255} \wedge \boldsymbol{\mu}_s[1] = -1 \\ \mathbf{sgn}(\boldsymbol{\mu}_s[0] \div \boldsymbol{\mu}_s[1]) \lfloor \lvert \boldsymbol{\mu}_s[0] \div \boldsymbol{\mu}_s[1] \rvert \rfloor & \text{otherwise} \end{cases}$ Where all values are treated as two's complement signed 256-bit integers. Note the overflow semantic when -2^{255} is negated.
0x06	MOD	2	1	Modulo remainder operation. $\boldsymbol{\mu}'_s[0] \equiv \begin{cases} 0 & \text{if} \quad \boldsymbol{\mu}_s[1] = 0 \\ \boldsymbol{\mu}_s[0] \bmod \boldsymbol{\mu}_s[1] & \text{otherwise} \end{cases}$
0x07	SMOD	2	1	Signed modulo remainder operation. $\boldsymbol{\mu}'_s[0] \equiv \begin{cases} 0 & \text{if} \quad \boldsymbol{\mu}_s[1] = 0 \\ \mathbf{sgn}(\boldsymbol{\mu}_s[0])(\lvert \boldsymbol{\mu}_s[0] \rvert \bmod \lvert \boldsymbol{\mu}_s[1] \rvert) & \text{otherwise} \end{cases}$ Where all values are treated as two's complement signed 256-bit integers.
0x08	ADDMOD	3	1	Modulo addition operation. $\boldsymbol{\mu}'_s[0] \equiv \begin{cases} 0 & \text{if} \quad \boldsymbol{\mu}_s[2] = 0 \\ (\boldsymbol{\mu}_s[0] + \boldsymbol{\mu}_s[1]) \mod \boldsymbol{\mu}_s[2] & \text{otherwise} \end{cases}$ All intermediate calculations of this operation are not subject to the 2^{256} modulo.
0x09	MULMOD	3	1	Modulo multiplication operation. $\boldsymbol{\mu}'_s[0] \equiv \begin{cases} 0 & \text{if} \quad \boldsymbol{\mu}_s[2] = 0 \\ (\boldsymbol{\mu}_s[0] \times \boldsymbol{\mu}_s[1]) \mod \boldsymbol{\mu}_s[2] & \text{otherwise} \end{cases}$ All intermediate calculations of this operation are not subject to the 2^{256} modulo.
0x0a	EXP	2	1	Exponential operation. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{\mu}_s[0]^{\boldsymbol{\mu}_s[1]}$
0x0b	SIGNEXTEND	2	1	Extend length of two's complement signed integer. $\forall i \in [0..255] : \boldsymbol{\mu}'_s[0]_i \equiv \begin{cases} \boldsymbol{\mu}_s[1]_t & \text{if} \quad i \leqslant t \quad \text{where } t = 256 - 8(\boldsymbol{\mu}_s[0] + 1) \\ \boldsymbol{\mu}_s[1]_i & \text{otherwise} \end{cases}$

$\boldsymbol{\mu}_s[x]_i$ gives the ith bit (counting from zero) of $\boldsymbol{\mu}_s[x]$

10s: Comparison & Bitwise Logic Operations

Value	Mnemonic	δ	α	Description
0x10	LT	2	1	Less-than comparison. $\boldsymbol{\mu}'_s[0] \equiv \begin{cases} 1 & \text{if} \quad \boldsymbol{\mu}_s[0] < \boldsymbol{\mu}_s[1] \\ 0 & \text{otherwise} \end{cases}$
0x11	GT	2	1	Greater-than comparison. $\boldsymbol{\mu}'_s[0] \equiv \begin{cases} 1 & \text{if} \quad \boldsymbol{\mu}_s[0] > \boldsymbol{\mu}_s[1] \\ 0 & \text{otherwise} \end{cases}$
0x12	SLT	2	1	Signed less-than comparison. $\boldsymbol{\mu}'_s[0] \equiv \begin{cases} 1 & \text{if} \quad \boldsymbol{\mu}_s[0] < \boldsymbol{\mu}_s[1] \\ 0 & \text{otherwise} \end{cases}$ Where all values are treated as two's complement signed 256-bit integers.
0x13	SGT	2	1	Signed greater-than comparison. $\boldsymbol{\mu}'_s[0] \equiv \begin{cases} 1 & \text{if} \quad \boldsymbol{\mu}_s[0] > \boldsymbol{\mu}_s[1] \\ 0 & \text{otherwise} \end{cases}$ Where all values are treated as two's complement signed 256-bit integers.
0x14	EQ	2	1	Equality comparison. $\boldsymbol{\mu}'_s[0] \equiv \begin{cases} 1 & \text{if} \quad \boldsymbol{\mu}_s[0] = \boldsymbol{\mu}_s[1] \\ 0 & \text{otherwise} \end{cases}$
0x15	ISZERO	1	1	Simple not operator. $\boldsymbol{\mu}'_s[0] \equiv \begin{cases} 1 & \text{if} \quad \boldsymbol{\mu}_s[0] = 0 \\ 0 & \text{otherwise} \end{cases}$
0x16	AND	2	1	Bitwise AND operation. $\forall i \in [0..255] : \boldsymbol{\mu}'_s[0]_i \equiv \boldsymbol{\mu}_s[0]_i \wedge \boldsymbol{\mu}_s[1]_i$
0x17	OR	2	1	Bitwise OR operation. $\forall i \in [0..255] : \boldsymbol{\mu}'_s[0]_i \equiv \boldsymbol{\mu}_s[0]_i \vee \boldsymbol{\mu}_s[1]_i$
0x18	XOR	2	1	Bitwise XOR operation. $\forall i \in [0..255] : \boldsymbol{\mu}'_s[0]_i \equiv \boldsymbol{\mu}_s[0]_i \oplus \boldsymbol{\mu}_s[1]_i$
0x19	NOT	1	1	Bitwise NOT operation. $\forall i \in [0..255] : \boldsymbol{\mu}'_s[0]_i \equiv \begin{cases} 1 & \text{if} \quad \boldsymbol{\mu}_s[0]_i = 0 \\ 0 & \text{otherwise} \end{cases}$
0x1a	BYTE	2	1	Retrieve single byte from word. $\forall i \in [0..255] : \boldsymbol{\mu}'_s[0]_i \equiv \begin{cases} \boldsymbol{\mu}_s[1]_{(i + 8\boldsymbol{\mu}_s[0])} & \text{if} \quad i < 8 \wedge \boldsymbol{\mu}_s[0] < 32 \\ 0 & \text{otherwise} \end{cases}$ For the Nth byte, we count from the left (i.e. N=0 would be the most significant in big endian).

20s: SHA3

Value	Mnemonic	δ	α	Description
0x20	SHA3	2	1	Compute Keccak-256 hash. $\boldsymbol{\mu}'_s[0] \equiv \mathtt{Keccak}(\boldsymbol{\mu}_m[\boldsymbol{\mu}_s[0] \dots (\boldsymbol{\mu}_s[0] + \boldsymbol{\mu}_s[1] - 1)])$ $\boldsymbol{\mu}'_i \equiv M(\boldsymbol{\mu}_i, \boldsymbol{\mu}_s[0], \boldsymbol{\mu}_s[1])$

30s: Environmental Information

Value	Mnemonic	δ	α	Description
0x30	ADDRESS	0	1	Get address of currently executing account. $\boldsymbol{\mu}'_s[0] \equiv I_a$
0x31	BALANCE	1	1	Get balance of the given account. $\boldsymbol{\mu}'_s[0] \equiv \begin{cases} \boldsymbol{\sigma}[\boldsymbol{\mu}_s[0]]_b & \text{if} \quad \boldsymbol{\sigma}[\boldsymbol{\mu}_s[0] \mod 2^{160}] \neq \varnothing \\ 0 & \text{otherwise} \end{cases}$
0x32	ORIGIN	0	1	Get execution origination address. $\boldsymbol{\mu}'_s[0] \equiv I_o$ This is the sender of original transaction; it is never an account with non-empty associated code.
0x33	CALLER	0	1	Get caller address. $\boldsymbol{\mu}'_s[0] \equiv I_s$ This is the address of the account that is directly responsible for this execution.
0x34	CALLVALUE	0	1	Get deposited value by the instruction/transaction responsible for this execution. $\boldsymbol{\mu}'_s[0] \equiv I_v$
0x35	CALLDATALOAD	1	1	Get input data of current environment. $\boldsymbol{\mu}'_s[0] \equiv I_{\mathbf{d}}[\boldsymbol{\mu}_s[0] \dots (\boldsymbol{\mu}_s[0] + 31)] \quad \text{with} \quad I_{\mathbf{d}}[x] = 0 \quad \text{if} \quad x \geqslant \lVert I_{\mathbf{d}} \rVert$ This pertains to the input data passed with the message call instruction or transaction.
0x36	CALLDATASIZE	0	1	Get size of input data in current environment. $\boldsymbol{\mu}'_s[0] \equiv \lVert I_{\mathbf{d}} \rVert$ This pertains to the input data passed with the message call instruction or transaction.
0x37	CALLDATACOPY	3	0	Copy input data in current environment to memory. $\forall_{i \in \{0 \dots \boldsymbol{\mu}_s[2]-1\}} \boldsymbol{\mu}'_m[\boldsymbol{\mu}_s[0] + i] \equiv \begin{cases} I_{\mathbf{d}}[\boldsymbol{\mu}_s[1] + i] & \text{if} \quad \boldsymbol{\mu}_s[1] + i < \lVert I_{\mathbf{d}} \rVert \\ 0 & \text{otherwise} \end{cases}$ The additions in $\boldsymbol{\mu}_s[1] + i$ are not subject to the 2^{256} modulo. $\boldsymbol{\mu}'_i \equiv M(\boldsymbol{\mu}_i, \boldsymbol{\mu}_s[0], \boldsymbol{\mu}_s[2])$ This pertains to the input data passed with the message call instruction or transaction.
0x38	CODESIZE	0	1	Get size of code running in current environment. $\boldsymbol{\mu}'_s[0] \equiv \lVert I_{\mathbf{b}} \rVert$
0x39	CODECOPY	3	0	Copy code running in current environment to memory. $\forall_{i \in \{0 \dots \boldsymbol{\mu}_s[2]-1\}} \boldsymbol{\mu}'_m[\boldsymbol{\mu}_s[0] + i] \equiv \begin{cases} I_{\mathbf{b}}[\boldsymbol{\mu}_s[1] + i] & \text{if} \quad \boldsymbol{\mu}_s[1] + i < \lVert I_{\mathbf{b}} \rVert \\ \text{STOP} & \text{otherwise} \end{cases}$ $\boldsymbol{\mu}'_i \equiv M(\boldsymbol{\mu}_i, \boldsymbol{\mu}_s[0], \boldsymbol{\mu}_s[2])$ The additions in $\boldsymbol{\mu}_s[1] + i$ are not subject to the 2^{256} modulo.
0x3a	GASPRICE	0	1	Get price of gas in current environment. $\boldsymbol{\mu}'_s[0] \equiv I_p$ This is gas price specified by the originating transaction.
0x3b	EXTCODESIZE	1	1	Get size of an account's code. $\boldsymbol{\mu}'_s[0] \equiv \lVert \boldsymbol{\sigma}[\boldsymbol{\mu}_s[0] \mod 2^{160}]_c \rVert$
0x3c	EXTCODECOPY	4	0	Copy an account's code to memory. $\forall_{i \in \{0 \dots \boldsymbol{\mu}_s[3]-1\}} \boldsymbol{\mu}'_m[\boldsymbol{\mu}_s[1] + i] \equiv \begin{cases} \mathbf{c}[\boldsymbol{\mu}_s[2] + i] & \text{if} \quad \boldsymbol{\mu}_s[2] + i < \lVert \mathbf{c} \rVert \\ \text{STOP} & \text{otherwise} \end{cases}$ where $\mathbf{c} \equiv \boldsymbol{\sigma}[\boldsymbol{\mu}_s[0] \mod 2^{160}]_c$ $\boldsymbol{\mu}'_i \equiv M(\boldsymbol{\mu}_i, \boldsymbol{\mu}_s[1], \boldsymbol{\mu}_s[3])$ The additions in $\boldsymbol{\mu}_s[2] + i$ are not subject to the 2^{256} modulo.
0x3d	RETURNDATASIZE	0	1	Get size of output data from the previous call from the current environment. $\boldsymbol{\mu}'_s[0] \equiv \lVert \boldsymbol{\mu}_o \rVert$
0x3e	RETURNDATACOPY	3	0	Copy output data from the previous call to memory. $\forall_{i \in \{0 \dots \boldsymbol{\mu}_s[2]-1\}} \boldsymbol{\mu}'_m[\boldsymbol{\mu}_s[0] + i] \equiv \begin{cases} \boldsymbol{\mu}_o[\boldsymbol{\mu}_s[1] + i] & \text{if} \quad \boldsymbol{\mu}_s[1] + i < \lVert \boldsymbol{\mu}_o \rVert \\ 0 & \text{otherwise} \end{cases}$ The additions in $\boldsymbol{\mu}_s[1] + i$ are not subject to the 2^{256} modulo. $\boldsymbol{\mu}'_i \equiv M(\boldsymbol{\mu}_i, \boldsymbol{\mu}_s[0], \boldsymbol{\mu}_s[2])$

40s: Block Information

Value	Mnemonic	δ	α	Description
0x40	BLOCKHASH	1	1	Get the hash of one of the 256 most recent complete blocks. $\boldsymbol{\mu}'_s[0] \equiv P(I_{H_p}, \boldsymbol{\mu}_s[0], 0)$ where P is the hash of a block of a particular number, up to a maximum age. 0 is left on the stack if the looked for block number is greater than the current block number or more than 256 blocks behind the current block. $P(h, n, a) \equiv \begin{cases} 0 & \text{if} \quad n > H_i \vee a = 256 \vee h = 0 \\ h & \text{if} \quad n = H_i \\ P(H_p, n, a + 1) & \text{otherwise} \end{cases}$ and we assert the header H can be determined from its hash h unless as its hash is the parent h is zero.
0x41	COINBASE	0	1	Get the block's beneficiary address. $\boldsymbol{\mu}'_s[0] \equiv I_{H_c}$
0x42	TIMESTAMP	0	1	Get the block's timestamp. $\boldsymbol{\mu}'_s[0] \equiv I_{H_s}$
0x43	NUMBER	0	1	Get the block's number. $\boldsymbol{\mu}'_s[0] \equiv I_{H_i}$
0x44	DIFFICULTY	0	1	Get the block's difficulty. $\boldsymbol{\mu}'_s[0] \equiv I_{H_d}$
0x45	GASLIMIT	0	1	Get the block's gas limit. $\boldsymbol{\mu}'_s[0] \equiv I_{H_l}$

50s: Stack, Memory, Storage and Flow Operations

Value	Mnemonic	δ	α	Description
0x50	POP	1	0	Remove item from stack.
0x51	MLOAD	1	1	Load word from memory. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{\mu}_m[\boldsymbol{\mu}_s[0] \dots (\boldsymbol{\mu}_s[0] + 31)]$ $\boldsymbol{\mu}'_i \equiv \max(\boldsymbol{\mu}_i, \lceil (\boldsymbol{\mu}_s[0] + 32) \div 32 \rceil)$ The addition in the calculation of $\boldsymbol{\mu}'_i$ is not subject to the 2^{256} modulo.
0x52	MSTORE	2	0	Save word to memory. $\boldsymbol{\mu}'_m[\boldsymbol{\mu}_s[0] \dots (\boldsymbol{\mu}_s[0] + 31)] \equiv \boldsymbol{\mu}_s[1]$ $\boldsymbol{\mu}'_i \equiv \max(\boldsymbol{\mu}_i, \lceil (\boldsymbol{\mu}_s[0] + 32) \div 32 \rceil)$ The addition in the calculation of $\boldsymbol{\mu}'_i$ is not subject to the 2^{256} modulo.
0x53	MSTORE8	2	0	Save byte to memory. $\boldsymbol{\mu}'_m[\boldsymbol{\mu}_s[0]] \equiv (\boldsymbol{\mu}_s[1] \bmod 256)$ $\boldsymbol{\mu}'_i \equiv \max(\boldsymbol{\mu}_i, \lceil (\boldsymbol{\mu}_s[0] + 1) \div 32 \rceil)$ The addition in the calculation of $\boldsymbol{\mu}'_i$ is not subject to the 2^{256} modulo.
0x54	SLOAD	1	1	Load word from storage. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{\sigma}[I_a]_s[\boldsymbol{\mu}_s[0]]$
0x55	SSTORE	2	0	Save word to storage. $\boldsymbol{\sigma}'[I_a]_s[\boldsymbol{\mu}_s[0]] \equiv \boldsymbol{\mu}_s[1]$ $C_{\text{SSTORE}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \equiv \begin{cases} G_{sset} & \text{if} \quad \boldsymbol{\mu}_s[1] \neq 0 \ \wedge \ \boldsymbol{\sigma}[I_a]_s[\boldsymbol{\mu}_s[0]] = 0 \\ G_{sreset} & \text{otherwise} \end{cases}$ $A'_r \equiv A_r + \begin{cases} R_{sclear} & \text{if} \quad \boldsymbol{\mu}_s[1] = 0 \ \wedge \ \boldsymbol{\sigma}[I_a]_s[\boldsymbol{\mu}_s[0]] \neq 0 \\ 0 & \text{otherwise} \end{cases}$
0x56	JUMP	1	0	Alter the program counter. $J_{\text{JUMP}}(\boldsymbol{\mu}) \equiv \boldsymbol{\mu}_s[0]$ This has the effect of writing said value to $\boldsymbol{\mu}_{pc}$. See section 9.
0x57	JUMPI	2	0	Conditionally alter the program counter. $J_{\text{JUMPI}}(\boldsymbol{\mu}) \equiv \begin{cases} \boldsymbol{\mu}_s[0] & \text{if} \quad \boldsymbol{\mu}_s[1] \neq 0 \\ \boldsymbol{\mu}_{pc} + 1 & \text{otherwise} \end{cases}$ This has the effect of writing said value to $\boldsymbol{\mu}_{pc}$. See section 9.
0x58	PC	0	1	Get the value of the program counter *prior* to the increment corresponding to this instruction. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{\mu}_{pc}$
0x59	MSIZE	0	1	Get the size of active memory in bytes. $\boldsymbol{\mu}'_s[0] \equiv 32\boldsymbol{\mu}_i$
0x5a	GAS	0	1	Get the amount of available gas, including the corresponding reduction for the cost of this instruction. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{\mu}_g$
0x5b	JUMPDEST	0	0	Mark a valid destination for jumps. This operation has no effect on machine state during execution.

60s & 70s: Push Operations

Value	Mnemonic	δ	α	Description
0x60	PUSH1	0	1	Place 1 byte item on stack. $\boldsymbol{\mu}'_s[0] \equiv c(\boldsymbol{\mu}_{pc} + 1)$ where $\quad c(x) \equiv \begin{cases} I_{\mathbf{b}}[x] & \text{if} \quad x < \lVert I_{\mathbf{b}} \rVert \\ 0 & \text{otherwise} \end{cases}$ The bytes are read in line from the program code's bytes array. The function c ensures the bytes default to zero if they extend past the limits. The byte is right-aligned (takes the lowest significant place in big endian).
0x61	PUSH2	0	1	Place 2-byte item on stack. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{c}\big((\boldsymbol{\mu}_{pc} + 1) \dots (\boldsymbol{\mu}_{pc} + 2)\big)$ with $\boldsymbol{c}(\boldsymbol{x}) \equiv (c(\boldsymbol{x}_0), \dots, c(\boldsymbol{x}_{\lVert \boldsymbol{x} \rVert - 1}))$ with c as defined as above. The bytes are right-aligned (takes the lowest significant place in big endian).
⋮	⋮	⋮	⋮	⋮
0x7f	PUSH32	0	1	Place 32-byte (full word) item on stack. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{c}\big((\boldsymbol{\mu}_{pc} + 1) \dots (\boldsymbol{\mu}_{pc} + 32)\big)$ where $\boldsymbol{c}$ is defined as above. The bytes are right-aligned (takes the lowest significant place in big endian).

80s: Duplication Operations

Value	Mnemonic	δ	α	Description
0x80	DUP1	1	2	Duplicate 1st stack item. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{\mu}_s[0]$
0x81	DUP2	2	3	Duplicate 2nd stack item. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{\mu}_s[1]$
⋮	⋮	⋮	⋮	⋮
0x8f	DUP16	16	17	Duplicate 16th stack item. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{\mu}_s[15]$

90s: Exchange Operations

Value	Mnemonic	δ	α	Description
0x90	SWAP1	2	2	Exchange 1st and 2nd stack items. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{\mu}_s[1]$ $\boldsymbol{\mu}'_s[1] \equiv \boldsymbol{\mu}_s[0]$
0x91	SWAP2	3	3	Exchange 1st and 3rd stack items. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{\mu}_s[2]$ $\boldsymbol{\mu}'_s[2] \equiv \boldsymbol{\mu}_s[0]$
⋮	⋮	⋮	⋮	⋮
0x9f	SWAP16	17	17	Exchange 1st and 17th stack items. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{\mu}_s[16]$ $\boldsymbol{\mu}'_s[16] \equiv \boldsymbol{\mu}_s[0]$

a0s: Logging Operations

For all logging operations, the state change is to append an additional log entry on to the substate's log series:

$A'_{\mathbf{l}} \equiv A_{\mathbf{l}} \cdot (I_a, \mathbf{t}, \boldsymbol{\mu}_m[\boldsymbol{\mu}_s[0] \dots (\boldsymbol{\mu}_s[0] + \boldsymbol{\mu}_s[1] - 1)])$

and to update the memory consumption counter:

$\boldsymbol{\mu}'_i \equiv M(\boldsymbol{\mu}_i, \boldsymbol{\mu}_s[0], \boldsymbol{\mu}_s[1])$

The entry's topic series, $\mathbf{t}$, differs accordingly:

Value	Mnemonic	δ	α	Description
0xa0	LOG0	2	0	Append log record with no topics. $\mathbf{t} \equiv ()$
0xa1	LOG1	3	0	Append log record with one topic. $\mathbf{t} \equiv (\boldsymbol{\mu}_s[2])$
⋮	⋮	⋮	⋮	⋮
0xa4	LOG4	6	0	Append log record with four topics. $\mathbf{t} \equiv (\boldsymbol{\mu}_s[2], \boldsymbol{\mu}_s[3], \boldsymbol{\mu}_s[4], \boldsymbol{\mu}_s[5])$

f0s: System operations

Value	Mnemonic	δ	α	Description
0xf0	CREATE	3	1	Create a new account with associated code. $\mathbf{i} \equiv \boldsymbol{\mu}_m[\boldsymbol{\mu}_s[1] \dots (\boldsymbol{\mu}_s[1] + \boldsymbol{\mu}_s[2] - 1)]$ $(\boldsymbol{\sigma}', \boldsymbol{\mu}'_g, A^+, \mathbf{o}) \equiv \begin{cases} \Lambda(\boldsymbol{\sigma}^*, I_a, I_o, L(\boldsymbol{\mu}_g), I_p, \boldsymbol{\mu}_s[0], \mathbf{i}, I_e + 1, I_w) & \text{if} \quad \boldsymbol{\mu}_s[0] \leqslant \boldsymbol{\sigma}[I_a]_b \\ & \wedge \ I_e < 1024 \\ (\boldsymbol{\sigma}, \boldsymbol{\mu}_g, \varnothing) & \text{otherwise} \end{cases}$ $\boldsymbol{\sigma}^* \equiv \boldsymbol{\sigma} \quad \text{except} \quad \boldsymbol{\sigma}^*[I_a]_n = \boldsymbol{\sigma}[I_a]_n + 1$ $A' \equiv A \Cup A^+$ which abbreviates: $A'_{\mathbf{s}} \equiv A_{\mathbf{s}} \cup A^+_{\mathbf{s}} \quad \wedge \quad A'_{\mathbf{l}} \equiv A_{\mathbf{l}} \cdot A^+_{\mathbf{l}} \quad \wedge$ $A'_{\mathbf{t}} \equiv A_{\mathbf{t}} \cup A^+_{\mathbf{t}} \wedge \quad A'_r \equiv A_r + A^+_r$ $\boldsymbol{\mu}'_s[0] \equiv x$ where $x = 0$ if the code execution for this operation failed due to an exceptional halting (or for a REVERT) $\boldsymbol{\sigma}' = \varnothing$, or $I_e = 1024$ (the maximum call depth limit is reached) or $\boldsymbol{\mu}_s[0] > \boldsymbol{\sigma}[I_a]_b$ (balance of the caller is too low to fulfil the value transfer); and otherwise $x = A(I_a, \boldsymbol{\sigma}[I_a]_n)$, the address of the newly created account. $\boldsymbol{\mu}'_i \equiv M(\boldsymbol{\mu}_i, \boldsymbol{\mu}_s[1], \boldsymbol{\mu}_s[2])$ $\boldsymbol{\mu}'_o \equiv ()$ Thus the operand order is: value, input offset, input size.
0xf1	CALL	7	1	Message-call into an account. $\mathbf{i} \equiv \boldsymbol{\mu}_m[\boldsymbol{\mu}_s[3] \dots (\boldsymbol{\mu}_s[3] + \boldsymbol{\mu}_s[4] - 1)]$ $(\boldsymbol{\sigma}', g', A^+, \mathbf{o}) \equiv \begin{cases} \Theta(\boldsymbol{\sigma}, I_a, I_o, t, t, C_{\text{CALLGAS}}(\boldsymbol{\mu}), & \text{if} \quad \boldsymbol{\mu}_s[2] \leqslant \boldsymbol{\sigma}[I_a]_b \ \wedge \\ \quad I_p, \boldsymbol{\mu}_s[2], \boldsymbol{\mu}_s[2], \mathbf{i}, I_e + 1, I_w) & \quad I_e < 1024 \\ (\boldsymbol{\sigma}, g, \varnothing, ()) & \text{otherwise} \end{cases}$ $n \equiv \min(\{\boldsymbol{\mu}_s[6], \lVert \mathbf{o} \rVert\})$ $\boldsymbol{\mu}'_m[\boldsymbol{\mu}_s[5] \dots (\boldsymbol{\mu}_s[5] + n - 1)] = \mathbf{o}[0 \dots (n - 1)]$ $\boldsymbol{\mu}'_o = \mathbf{o}$ $\boldsymbol{\mu}'_g \equiv \boldsymbol{\mu}_g + g'$ $\boldsymbol{\mu}'_s[0] \equiv x$ $A' \equiv A \Cup A^+$ $t \equiv \boldsymbol{\mu}_s[1] \mod 2^{160}$ where $x = 0$ if the code execution for this operation failed due to an exceptional halting (or for a REVERT) $\boldsymbol{\sigma}' = \varnothing$ or if $\boldsymbol{\mu}_s[2] > \boldsymbol{\sigma}[I_a]_b$ (not enough funds) or $I_e = 1024$ (call depth limit reached); $x = 1$ otherwise. $\boldsymbol{\mu}'_i \equiv M(M(\boldsymbol{\mu}_i, \boldsymbol{\mu}_s[3], \boldsymbol{\mu}_s[4]), \boldsymbol{\mu}_s[5], \boldsymbol{\mu}_s[6])$ Thus the operand order is: gas, to, value, in offset, in size, out offset, out size. $C_{\text{CALL}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \equiv C_{\text{GASCAP}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) + C_{\text{EXTRA}}(\boldsymbol{\sigma}, \boldsymbol{\mu})$ $C_{\text{CALLGAS}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \equiv \begin{cases} C_{\text{GASCAP}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) + G_{callstipend} & \text{if} \quad \boldsymbol{\mu}_s[2] \neq 0 \\ C_{\text{GASCAP}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) & \text{otherwise} \end{cases}$ $C_{\text{GASCAP}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \equiv \begin{cases} \min\{L(\boldsymbol{\mu}_g - C_{\text{EXTRA}}(\boldsymbol{\sigma}, \boldsymbol{\mu})), \boldsymbol{\mu}_s[0]\} & \text{if} \quad \boldsymbol{\mu}_g \geq C_{\text{EXTRA}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \\ \boldsymbol{\mu}_s[0] & \text{otherwise} \end{cases}$ $C_{\text{EXTRA}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \equiv G_{call} + C_{\text{XFER}}(\boldsymbol{\mu}) + C_{\text{NEW}}(\boldsymbol{\sigma}, \boldsymbol{\mu})$ $C_{\text{XFER}}(\boldsymbol{\mu}) \equiv \begin{cases} G_{callvalue} & \text{if} \quad \boldsymbol{\mu}_s[2] \neq 0 \\ 0 & \text{otherwise} \end{cases}$ $C_{\text{NEW}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \equiv \begin{cases} G_{newaccount} & \text{if} \quad \mathtt{DEAD}(\boldsymbol{\sigma}, \boldsymbol{\mu}_s[1] \mod 2^{160}) \wedge \boldsymbol{\mu}_s[2] \neq 0 \\ 0 & \text{otherwise} \end{cases}$
0xf2	CALLCODE	7	1	Message-call into this account with an alternative account's code. Exactly equivalent to CALL except: $(\boldsymbol{\sigma}', g', A^+, \mathbf{o}) \equiv \begin{cases} \Theta(\boldsymbol{\sigma}^*, I_a, I_o, I_a, t, C_{\text{CALLGAS}}(\boldsymbol{\mu}), & \text{if} \quad \boldsymbol{\mu}_s[2] \leqslant \boldsymbol{\sigma}[I_a]_b \ \wedge \\ \quad I_p, \boldsymbol{\mu}_s[2], \boldsymbol{\mu}_s[2], \mathbf{i}, I_e + 1, I_w) & \quad I_e < 1024 \\ (\boldsymbol{\sigma}, g, \varnothing, ()) & \text{otherwise} \end{cases}$ Note the change in the fourth parameter to the call Θ from the 2nd stack value $\boldsymbol{\mu}_s[1]$ (as in CALL) to the present address I_a. This means that the recipient is in fact the same account as at present, simply that the code is overwritten.
0xf3	RETURN	2	0	Halt execution returning output data. $H_{\text{RETURN}}(\boldsymbol{\mu}) \equiv \boldsymbol{\mu}_m[\boldsymbol{\mu}_s[0] \dots (\boldsymbol{\mu}_s[0] + \boldsymbol{\mu}_s[1] - 1)]$ This has the effect of halting the execution at this point with output defined. See section 9. $\boldsymbol{\mu}'_i \equiv M(\boldsymbol{\mu}_i, \boldsymbol{\mu}_s[0], \boldsymbol{\mu}_s[1])$

0xf4	DELEGATECALL	6	1	Message-call into this account with an alternative account's code, but persisting the current values for *sender* and *value*. Compared with CALL, DELEGATECALL takes one fewer arguments. The omitted argument is $\boldsymbol{\mu}_s[2]$. As a result, $\boldsymbol{\mu}_s[3]$, $\boldsymbol{\mu}_s[4]$, $\boldsymbol{\mu}_s[5]$ and $\boldsymbol{\mu}_s[6]$ in the definition of CALL should respectively be replaced with $\boldsymbol{\mu}_s[2]$, $\boldsymbol{\mu}_s[3]$, $\boldsymbol{\mu}_s[4]$ and $\boldsymbol{\mu}_s[5]$. Otherwise it is equivalent to CALL except: $(\boldsymbol{\sigma}', g', A^+, \mathbf{o}) \equiv \begin{cases} \Theta(\boldsymbol{\sigma}^*, I_s, I_o, I_a, t, & \text{if} \quad I_v \leqslant \boldsymbol{\sigma}[I_s]_b \ \wedge \\ \quad \boldsymbol{\mu}_s[0], I_p, 0, I_v, \mathbf{i}, I_e + 1, I_w) & \\ I_e < 1024 & \\ (\boldsymbol{\sigma}, g, \varnothing, ()) & \text{otherwise} \end{cases}$ Note the changes (in addition to that of the fourth parameter) to the second and ninth parameters to the call Θ. This means that the recipient is in fact the same account as at present, simply that the code is overwritten *and* the context is almost entirely identical.
0xfa	STATICCALL	6	1	Static message-call into an account. Exactly equivalent to CALL except: The argument $\boldsymbol{\mu}_s[2]$ is replaced with 0. The deeper argument $\boldsymbol{\mu}_s[3]$, $\boldsymbol{\mu}_s[4]$, $\boldsymbol{\mu}_s[5]$ and $\boldsymbol{\mu}_s[6]$ are respectively replaced with $\boldsymbol{\mu}_s[2]$, $\boldsymbol{\mu}_s[3]$, $\boldsymbol{\mu}_s[4]$ and $\boldsymbol{\mu}_s[5]$. The last argument of Θ is $\bot$.
0xfd	REVERT	2	0	Halt execution reverting state changes but returning data and remaining gas. The effect of this operation is described in (131). For the gas calculation, we use the memory expansion function, $\boldsymbol{\mu}'_i \equiv M(\boldsymbol{\mu}_i, \boldsymbol{\mu}_s[0], \boldsymbol{\mu}_s[1])$
0xfe	INVALID	$\varnothing$	$\varnothing$	Designated invalid instruction.
0xff	SELFDESTRUCT	1	0	Halt execution and register account for later deletion. $A'_{\mathbf{s}} \equiv A_{\mathbf{s}} \cup \{I_a\}$ $\boldsymbol{\sigma}'[r] \equiv \begin{cases} \varnothing & \text{if } \boldsymbol{\sigma}[r] = \varnothing \ \wedge \ \boldsymbol{\sigma}[I_a]_b = 0 \\ (\boldsymbol{\sigma}[r]_n, \boldsymbol{\sigma}[r]_b + \boldsymbol{\sigma}[I_a]_b, \boldsymbol{\sigma}[r]_s, \boldsymbol{\sigma}[r]_c) & \text{if } r \neq I_a \\ (\boldsymbol{\sigma}[r]_n, 0, \boldsymbol{\sigma}[r]_s, \boldsymbol{\sigma}[r]_c) & \text{otherwise} \end{cases}$ where $r = \boldsymbol{\mu}_s[0] \bmod 2^{160}$ $\boldsymbol{\sigma}'[I_a]_b = 0$ $C_{\text{SELFDESTRUCT}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \equiv G_{selfdestruct} + \begin{cases} G_{newaccount} & \text{if} \quad n \\ 0 & \text{otherwise} \end{cases}$ $n \equiv \mathtt{DEAD}(\boldsymbol{\sigma}, \boldsymbol{\mu}_s[0] \mod 2^{160}) \wedge \boldsymbol{\sigma}[I_a]_b \neq 0$

Appendix I. Genesis Block

The genesis block is 15 items, and is specified thus:

$$\big((0_{256}, \mathtt{KEC}\big(\mathtt{RLP}\big(()\big)\big), 0_{160}, stateRoot, 0, 0, 0_{2048}, 2^{17}, 0, 0, 3141592, time, 0, 0_{256}, \mathtt{KEC}\big((42)\big)), (), ()\big) \tag{300}$$

Where 0_{256} refers to the parent hash, a 256-bit hash which is all zeroes; 0_{160} refers to the beneficiary address, a 160-bit hash which is all zeroes; 0_{2048} refers to the log bloom, 2048-bit of all zeros; 2^{17} refers to the difficulty; the transaction trie root, receipt trie root, gas used, block number and extradata are both 0, being equivalent to the empty byte array. The sequences of both ommers and transactions are empty and represented by (). $\mathtt{KEC}\big((42)\big)$ refers to the Keccak hash of a byte array of length one whose first and only byte is of value 42, used for the nonce. $\mathtt{KEC}\big(\mathtt{RLP}\big(()\big)\big)$ value refers to the hash of the ommer list in RLP, both empty lists.

The proof-of-concept series include a development premine, making the state root hash some value *stateRoot*. Also *time* will be set to the initial timestamp of the genesis block. The latest documentation should be consulted for those values.

Appendix J. Ethash

J.1. **Definitions.** We employ the following definitions:

Name	Value	Description
$J_{wordbytes}$	4	Bytes in word.
$J_{datasetinit}$	2^{30}	Bytes in dataset at genesis.
$J_{datasetgrowth}$	2^{23}	Dataset growth per epoch.
$J_{cacheinit}$	2^{24}	Bytes in cache at genesis.
$J_{cachegrowth}$	2^{17}	Cache growth per epoch.
J_{epoch}	30000	Blocks per epoch.
$J_{mixbytes}$	128	mix length in bytes.
$J_{hashbytes}$	64	Hash length in bytes.
$J_{parents}$	256	Number of parents of each dataset element.
$J_{cacherounds}$	3	Number of rounds in cache production.
$J_{accesses}$	64	Number of accesses in hashimoto loop.

J.2. **Size of dataset and cache.** The size for Ethash's cache $\mathbf{c} \in \mathbb{B}$ and dataset $\mathbf{d} \in \mathbb{B}$ depend on the epoch, which in turn depends on the block number.

$$E_{epoch}(H_i) = \left\lfloor \frac{H_i}{J_{epoch}} \right\rfloor \tag{301}$$

The size of the dataset growth by $J_{datasetgrowth}$ bytes, and the size of the cache by $J_{cachegrowth}$ bytes, every epoch. In order to avoid regularity leading to cyclic behavior, the size must be a prime number. Therefore the size is reduced by a multiple of $J_{mixbytes}$, for the dataset, and $J_{hashbytes}$ for the cache. Let $d_{size} = \lVert \mathbf{d} \rVert$ be the size of the dataset. Which is calculated using

$$d_{size} = E_{prime}(J_{datasetinit} + J_{datasetgrowth} \cdot E_{epoch} - J_{mixbytes}, J_{mixbytes}) \tag{302}$$

The size of the cache, c_{size}, is calculated using

$$c_{size} = E_{prime}(J_{cacheinit} + J_{cachegrowth} \cdot E_{epoch} - J_{hashbytes}, J_{hashbytes}) \tag{303}$$

$$E_{prime}(x, y) = \begin{cases} x & \text{if} \quad x/y \in \mathbb{N} \\ E_{prime}(x - 2 \cdot y, y) & \text{otherwise} \end{cases} \tag{304}$$

J.3. **Dataset generation.** In order to generate the dataset we need the cache $\mathbf{c}$, which is an array of bytes. It depends on the cache size c_{size} and the seed hash $\mathbf{s} \in \mathbb{B}_{32}$.

J.3.1. *Seed hash.* The seed hash is different for every epoch. For the first epoch it is the Keccak-256 hash of a series of 32 bytes of zeros. For every other epoch it is always the Keccak-256 hash of the previous seed hash:

$$\mathbf{s} = C_{seedhash}(H_i) \tag{305}$$

$$C_{seedhash}(H_i) = \begin{cases} \mathbf{0}_{32} & \text{if} \quad E_{epoch}(H_i) = 0 \\ \mathtt{KEC}(C_{seedhash}(H_i - J_{epoch})) & \text{otherwise} \end{cases} \tag{306}$$

With $\mathbf{0}_{32}$ being 32 bytes of zeros.

J.3.2. *Cache.* The cache production process involves using the seed hash to first sequentially filling up c_{size} bytes of memory, then performing $J_{cacherounds}$ passes of the RandMemoHash algorithm created by Lerner [2014]. The initial cache $\mathbf{c}'$, being an array of arrays of single bytes, will be constructed as follows.

We define the array $\mathbf{c}_i$, consisting of 64 single bytes, as the ith element of the initial cache:

$$\mathbf{c}_i = \begin{cases} \mathtt{KEC512}(\mathbf{s}) & \text{if} \quad i = 0 \\ \mathtt{KEC512}(\mathbf{c}_{i-1}) & \text{otherwise} \end{cases} \tag{307}$$

Therefore $\mathbf{c}'$ can be defined as

$$\mathbf{c}'[i] = \mathbf{c}_i \quad \forall \quad i < n \tag{308}$$

$$n = \left\lfloor \frac{c_{size}}{J_{hashbytes}} \right\rfloor \tag{309}$$

The cache is calculated by performing $J_{cacherounds}$ rounds of the RandMemoHash algorithm to the initial cache $\mathbf{c}'$:

$$\mathbf{c} = E_{cacherounds}(\mathbf{c}', J_{cacherounds}) \tag{310}$$

$$E_{cacherounds}(\mathbf{x}, y) = \begin{cases} \mathbf{x} & \text{if} \quad y = 0 \\ E_{\text{RMH}}(\mathbf{x}) & \text{if} \quad y = 1 \\ E_{cacherounds}(E_{\text{RMH}}(\mathbf{x}), y - 1) & \text{otherwise} \end{cases} \tag{311}$$

Where a single round modifies each subset of the cache as follows:

$$E_{\text{RMH}}(\mathbf{x}) = \big(E_{rmh}(\mathbf{x}, 0), E_{rmh}(\mathbf{x}, 1), ..., E_{rmh}(\mathbf{x}, n - 1)\big) \tag{312}$$

0xf1	CALL	7	1	Message-call into an account.

$\mathbf{i} \equiv \boldsymbol{\mu}_\mathbf{m}[\boldsymbol{\mu}_\mathbf{s}[3] \dots (\boldsymbol{\mu}_\mathbf{s}[3] + \boldsymbol{\mu}_\mathbf{s}[4] - 1)]$

$$(\boldsymbol{\sigma}', g', A^+, \mathbf{o}) \equiv \begin{cases} \Theta(\boldsymbol{\sigma}, I_\mathrm{a}, I_\mathrm{o}, t, t, C_{\mathrm{CALLGAS}}(\boldsymbol{\mu}), & \text{if} \quad \boldsymbol{\mu}_\mathbf{s}[2] \leqslant \boldsymbol{\sigma}[I_\mathrm{a}]_\mathrm{b} \wedge \\ \quad I_\mathrm{p}, \boldsymbol{\mu}_\mathbf{s}[2], \boldsymbol{\mu}_\mathbf{s}[2], \mathbf{i}, I_\mathrm{e} + 1, I_\mathrm{w}) & \quad I_\mathrm{e} < 1024 \\ (\boldsymbol{\sigma}, g, \varnothing, ()) & \text{otherwise} \end{cases}$$

$n \equiv \min(\{\boldsymbol{\mu}_\mathbf{s}[6], |\mathbf{o}|\})$

$\boldsymbol{\mu}'_\mathbf{m}[\boldsymbol{\mu}_\mathbf{s}[5] \dots (\boldsymbol{\mu}_\mathbf{s}[5] + n - 1)] = \mathbf{o}[0 \dots (n-1)]$

$\boldsymbol{\mu}'_\mathbf{o} = \mathbf{o}$

$\boldsymbol{\mu}'_\mathrm{g} \equiv \boldsymbol{\mu}_\mathrm{g} + g'$

$\boldsymbol{\mu}'_\mathbf{s}[0] \equiv x$

$A' \equiv A \uplus A^+$

$t \equiv \boldsymbol{\mu}_\mathbf{s}[1] \mod 2^{160}$

where $x = 0$ if the code execution for this operation failed due to an exceptional halting (or for a REVERT) $\boldsymbol{\sigma}' = \varnothing$ or if $\boldsymbol{\mu}_\mathbf{s}[2] > \boldsymbol{\sigma}[I_\mathrm{a}]_\mathrm{b}$ (not enough funds) or $I_\mathrm{e} = 1024$ (call depth limit reached); $x = 1$ otherwise.

$\boldsymbol{\mu}'_\mathrm{i} \equiv M(M(\boldsymbol{\mu}_\mathrm{i}, \boldsymbol{\mu}_\mathbf{s}[3], \boldsymbol{\mu}_\mathbf{s}[4]), \boldsymbol{\mu}_\mathbf{s}[5], \boldsymbol{\mu}_\mathbf{s}[6])$

Thus the operand order is: gas, to, value, in offset, in size, out offset, out size.

$$C_{\mathrm{CALL}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \equiv C_{\mathrm{GASCAP}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) + C_{\mathrm{EXTRA}}(\boldsymbol{\sigma}, \boldsymbol{\mu})$$

$$C_{\mathrm{CALLGAS}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \equiv \begin{cases} C_{\mathrm{GASCAP}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) + G_{callstipend} & \text{if} \quad \boldsymbol{\mu}_\mathbf{s}[2] \neq 0 \\ C_{\mathrm{GASCAP}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) & \text{otherwise} \end{cases}$$

$$C_{\mathrm{GASCAP}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \equiv \begin{cases} \min\{L(\boldsymbol{\mu}_\mathrm{g} - C_{\mathrm{EXTRA}}(\boldsymbol{\sigma}, \boldsymbol{\mu})), \boldsymbol{\mu}_\mathbf{s}[0]\} & \text{if} \quad \boldsymbol{\mu}_\mathrm{g} \geq C_{\mathrm{EXTRA}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \\ \boldsymbol{\mu}_\mathbf{s}[0] & \text{otherwise} \end{cases}$$

$$C_{\mathrm{EXTRA}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \equiv G_{call} + C_{\mathrm{XFER}}(\boldsymbol{\mu}) + C_{\mathrm{NEW}}(\boldsymbol{\sigma}, \boldsymbol{\mu})$$

$$C_{\mathrm{XFER}}(\boldsymbol{\mu}) \equiv \begin{cases} G_{callvalue} & \text{if} \quad \boldsymbol{\mu}_\mathbf{s}[2] \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$C_{\mathrm{NEW}}(\boldsymbol{\sigma}, \boldsymbol{\mu}) \equiv \begin{cases} G_{newaccount} & \text{if} \quad \mathtt{DEAD}(\boldsymbol{\sigma}, \boldsymbol{\mu}_\mathbf{s}[1] \mod 2^{160}) \wedge \boldsymbol{\mu}_\mathbf{s}[2] \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

- Smart Contracts werden von **allen Minern** ausgeführt

- Smart Contracts werden von **allen Minern** ausgeführt
- Berechnung der **Transaktionsgebühr** anhand der Ausführungsdauer

- Smart Contracts werden von **allen Minern** ausgeführt
- Berechnung der **Transaktionsgebühr** anhand der Ausführungsdauer
- User muss eine **Tankgröße** und einen **Spritpreis** angeben

- Smart Contracts werden von **allen Minern** ausgeführt
- Berechnung der **Transaktionsgebühr** anhand der Ausführungsdauer
- User muss eine **Tankgröße** und einen **Spritpreis** angeben
- jede Operation hat einen definierten **Verbrauch**

Send ETH

Only send ETH to an Ethereum address.

From:

Test
2.998131 ETH
$411.85 USD

To:

0x0a1c8acab8a47D27dA0e

Amount:
Max

1 ETH
$137.37 USD

Transaction Fee:

Slow	Average	Fast
0.00008 ETH	0.00021 ETH	0.00042 ETH
$0.01	$0.03	$0.06

Advanced Options

CANCEL

NEXT

Customize Gas

Close

Advanced

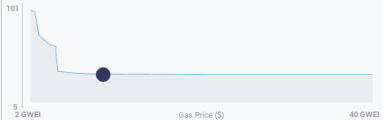
New Transaction Fee
0.00021 ETH

~Transaction Time
~31 sec

Gas Price (GWEI)
10

Gas Limit
21000

Live Gas Price Predictions



Send Amount	1 ETH
Transaction Fee	0.00021 ETH
New Total	**1.00021 ETH**
	$137.40

SAVE

```
pragma solidity >=0.4.22 <0.6.0;

contract MyToken {
    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;

    /* Initializes contract with initial supply tokens to the creator of the contract */
    constructor(
        uint256 initialSupply
        ) public {
        balanceOf[msg.sender] = initialSupply;              // Give the creator all initial tokens
    }

    /* Send coins */
    function transfer(address _to, uint256 _value) public returns (bool success) {
        require(balanceOf[msg.sender] >= _value);           // Check if the sender has enough
        require(balanceOf[_to] + _value >= balanceOf[_to]); // Check for overflows
        balanceOf[msg.sender] -= _value;                    // Subtract from the sender
        balanceOf[_to] += _value;                           // Add the same to the recipient
        return true;
```

Programmierung & Tools

Ein einfacher Contract

```
608060405234801561001057600080fd5b506000808190555033600160006101000a81548173ffff
ffffffffffffffffffffffffffffffffffffffff02191690837 3ffffffffffffffffffffffffffffff
ffffffff1602179055506101be80610068600039600 0f3006080604052600436106100 4c57600035
7c0100000000000000000000000000000000000000000000000000000000900463ffffffff168063
3ccfd60b146100515780638f9595d414610068575b600080fd5b34801561005d57600080fd5b5061
0066610086565b005b61007061011d565b604051808281526020019150506040518091039 0f35b60
0160009054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffff
ffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffffffffffffffffff1614
15156100e257600080fd5b6001600090549061010 00a900473ffffffffffffffffffffffffffffff
ffffffffff1673ffffffffffffffffffffffffffffffffffffffff16ff5b6000600160009054906 1
01000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffff
ffffffffffffffff163373ffffffffffffffffffffffffffffffffffffffff1614151561017b5760
0080fd5b34600080828254019250508190555060005490509056 00a165627a7a72305820b4d2d5a2
acfa0e197908c7c57826d35a987f64b29f8b97020a4ca96f63ce66630029
```

Ein einfacher Contract

Solidity

- populärste Hochsprache für EVM-Bytecode
- getypt, objektorientiert
- syntaktisch an Javascript angelehnt

```
pragma solidity ^0.4.22;

contract Wallet {

    uint256 balance;
    address owner;

    constructor () public {
        balance = 0;
        owner = msg.sender;
    }

    function addfund() payable public returns (uint256) {
        require (msg.sender == owner);
        balance += msg.value;
        return balance;
    }

    function withdraw() public {
        require (msg.sender == owner);
        selfdestruct(owner);
    }

}
```

```
pragma solidity ^0.4.22;

contract Wallet {

    uint256 balance;
    address owner;

    constructor () public {
        balance = 0;
        owner = msg.sender;
    }

    function addfund() payable public returns (uint256) {
        require (msg.sender == owner);
        balance += msg.value;
        return balance;
    }

    function withdraw() public {
        require (msg.sender == owner);
        selfdestruct(owner);
    }

}
```

```
pragma solidity ^0.4.22;

contract Wallet {

    uint256 balance;
    address owner;

    constructor () public {
        balance = 0;
        owner = msg.sender;
    }

    function addfund() payable public returns (uint256) {
        require (msg.sender == owner);
        balance += msg.value;
        return balance;
    }

    function withdraw() public {
        require (msg.sender == owner);
        selfdestruct(owner);
    }

}
```

```
pragma solidity ^0.4.22;
contract Wallet {
    uint256 balance;
    address owner;
    constructor () public {
        balance = 0;
        owner = msg.sender;
    }
    function addfund() payable public returns (uint256) {
        require (msg.sender == owner);
        balance += msg.value;
        return balance;
    }
    function withdraw() public {
        require (msg.sender == owner);
        selfdestruct(owner);
    }
}
```

```
pragma solidity ^0.4.22;

contract Wallet {

    uint256 balance;
    address owner;

    constructor () public {
        balance = 0;
        owner = msg.sender;
    }

    function addfund() payable public returns (uint256) {
        require (msg.sender == owner);
        balance += msg.value;
        return balance;
    }

    function withdraw() public {
        require (msg.sender == owner);
        selfdestruct(owner);
    }

}
```

```
pragma solidity ^0.4.22;

contract Wallet {

    uint256 balance;
    address owner;

    constructor () public {
        balance = 0;
        owner = msg.sender;
    }

    function addfund() payable public returns (uint256) {
        require (msg.sender == owner);
        balance += msg.value;
        return balance;
    }

    function withdraw() public {
        require (msg.sender == owner);
        selfdestruct(owner);
    }

}
```

```
pragma solidity ^0.4.22;

contract Wallet {

    uint256 balance;
    address owner;

    constructor () public {
        balance = 0;
        owner = msg.sender;
    }

    function addfund() payable public returns (uint256) {
        require (msg.sender == owner);
        balance += msg.value;
        return balance;
    }

    function withdraw() public {
        require (msg.sender == owner);
        selfdestruct(owner);
    }

}
```

```
pragma solidity ^0.4.22;

contract Wallet {

    uint256 balance;
    address owner;

    constructor () public {
        balance = 0;
        owner = msg.sender;
    }

    function addfund() payable public returns (uint256) {
        require (msg.sender == owner);
        balance += msg.value;
        return balance;
    }

    function withdraw() public {
        require (msg.sender == owner);
        selfdestruct(owner);
    }

}
```

```
pragma solidity ^0.4.22;

contract Wallet {

    uint256 balance;
    address owner;

    constructor () public {
        balance = 0;
        owner = msg.sender;
    }

    function addfund() payable public returns (uint256) {
        require (msg.sender == owner);
        balance += msg.value;
        return balance;
    }

    function withdraw() public {
        require (msg.sender == owner);
        selfdestruct(owner);
    }

}
```

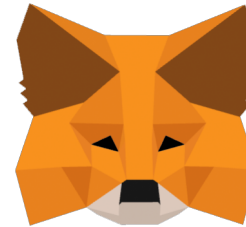
web3.js

- official JavaScript API for Ethereum
- available for the browser and Node.js

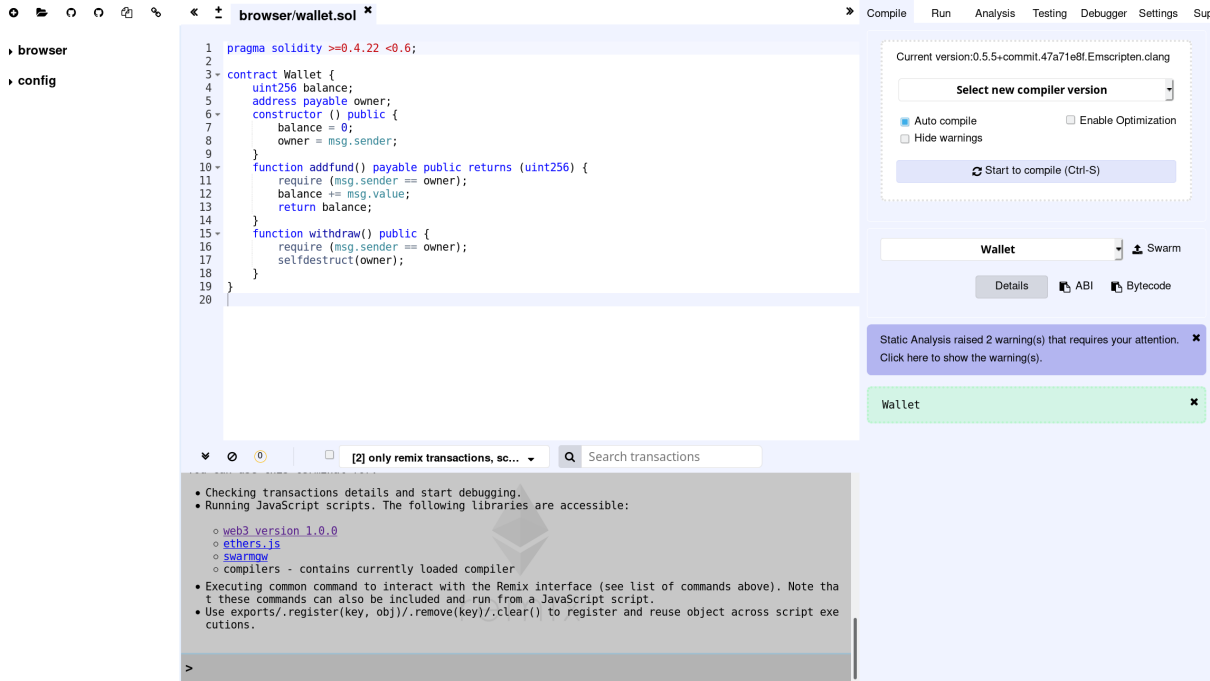
web3.js

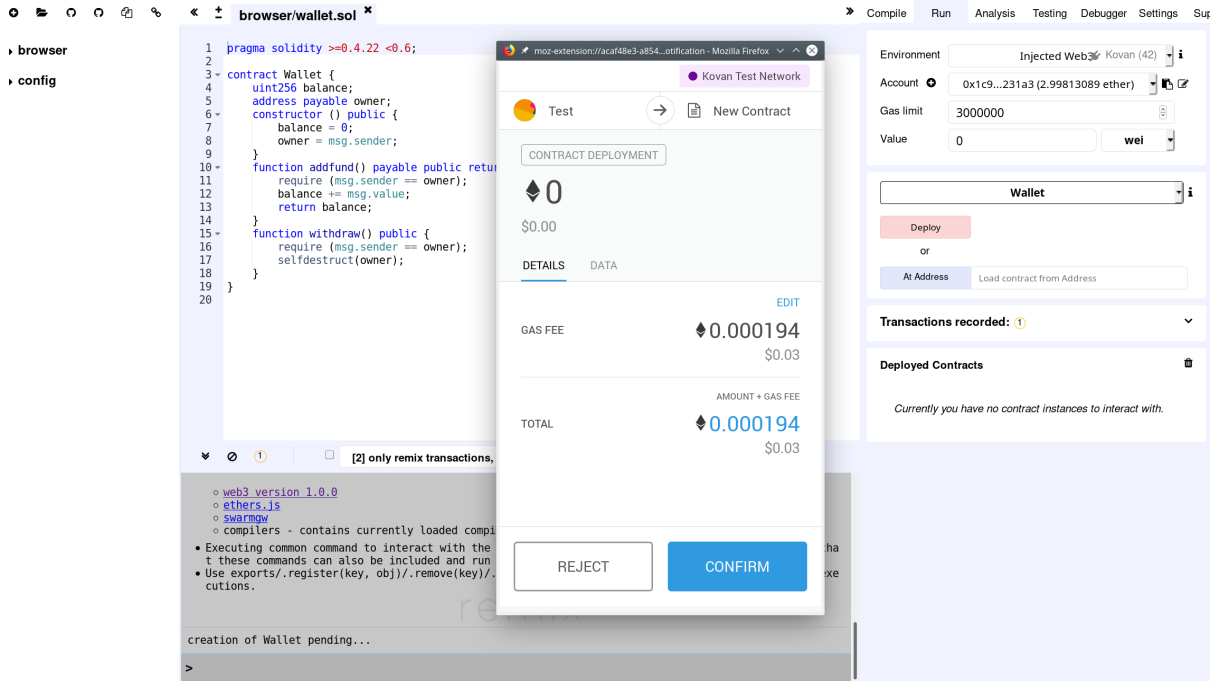
```
var contract = new web3.eth.Contract(
  abi,
  {data: codeHex, from: web3.eth.defaultAccount });
var deployTransaction = contract.deploy({data: codeHex, arguments: []});
web3.eth.personal.unlockAccount(web3.eth.defaultAccount, "user")
  .then(() => deployTransaction.estimateGas())
  .then(gas =>
    deployTransaction.send({gasLimit: gas, from: web3.eth.defaultAccount}))
  .then(contract =>
    console.log("Address of new contract: " + contract.options.address))
  .catch(err => console.log(err));
```

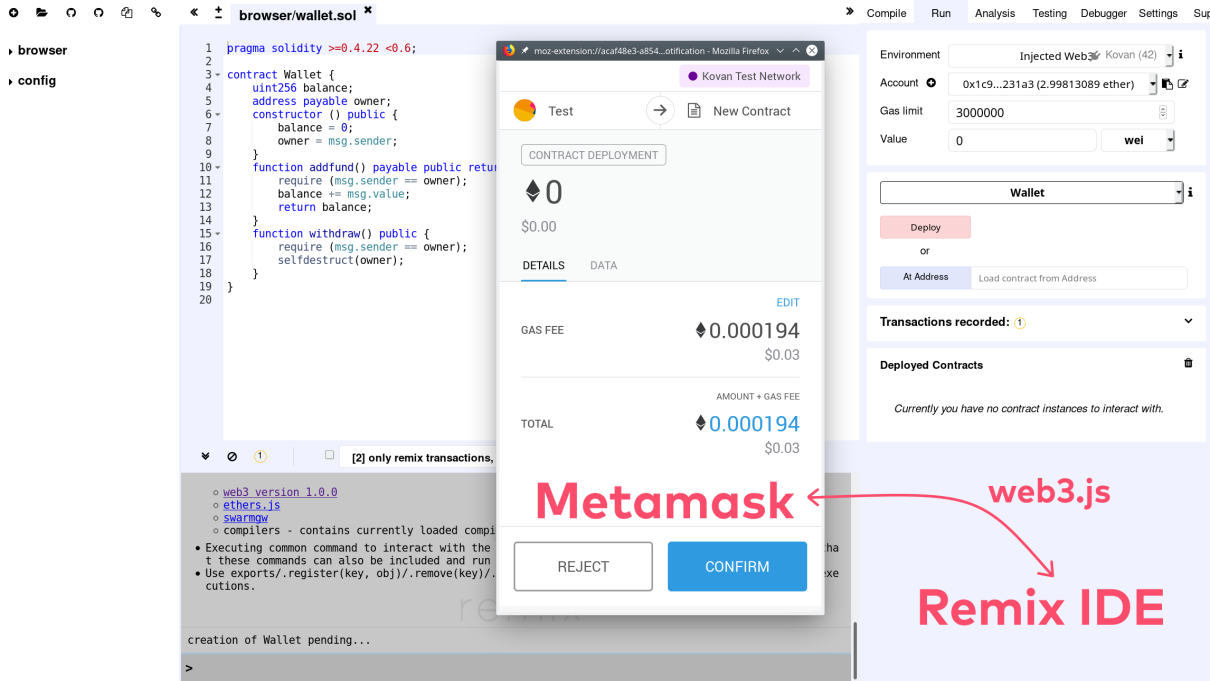
Metamask



- Browser-Plugin zur Verwaltung von Ethereum-Adressen
- kommunziert über RPC mit Blockchain-Knoten
- integriert mit web3.js


browser
config
browser/wallet.sol
Compile
Run
Analysis
Testing
Debugger
Settings
Sup
pragma solidity >=0.4.22 <0.6;
contract Wallet {
uint256 balance;
address payable owner;
constructor () public {
balance = 0;
owner = msg.sender;
}
function addfund() payable public returns (uint256) {
require (msg.sender == owner);
balance += msg.value;
return balance;
}
function withdraw() public {
require (msg.sender == owner);
selfdestruct(owner);
}
}
Current version:0.5.5+commit.47a71e8f.Emscripten.clang
Select new compiler version
Auto compile
Enable Optimization
Hide warnings
Start to compile (Ctrl-S)
Wallet
Swarm
Details
ABI
Bytecode
Static Analysis raised 2 warning(s) that requires your attention.
Click here to show the warning(s).
Wallet
[2] only remix transactions, sc...
Search transactions
• Checking transactions details and start debugging.
• Running JavaScript scripts. The following libraries are accessible:
web3 version 1.0.0
ethers.js
swarmgw
compilers - contains currently loaded compiler
• Executing common command to interact with the Remix interface (see list of commands above). Note tha t these commands can also be included and run from a JavaScript script.
• Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse object across script exe cutions.
>


browser
config
browser/wallet.sol
Compile
Run
Analysis
Testing
Debugger
Settings
pragma solidity >=0.4.22 <0.6;
contract Wallet {
uint256 balance;
address payable owner;
constructor () public {
balance = 0;
owner = msg.sender;
}
function addfund() payable public retur
require (msg.sender == owner);
balance += msg.value;
return balance;
}
function withdraw() public {
require (msg.sender == owner);
selfdestruct(owner);
}
}
[2] only remix transactions,
web3 version 1.0.0
ethers.js
swarmgw
compilers - contains currently loaded compi
Executing common command to interact with the
t these commands can also be included and run
Use exports/.register(key, obj)/.remove(key)/.
cutions.
creation of Wallet pending...
>
moz-extension://acaf48e3-a854...otification - Mozilla Firefox
Kovan Test Network
Test
New Contract
CONTRACT DEPLOYMENT
0
$0.00
DETAILS
DATA
EDIT
GAS FEE
0.000194
$0.03
AMOUNT + GAS FEE
TOTAL
0.000194
$0.03
REJECT
CONFIRM
Environment
Injected Web3
Kovan (42)
Account
0x1c9...231a3 (2.99813089 ether)
Gas limit
3000000
Value
0
wei
Wallet
Deploy
or
At Address
Load contract from Address
Transactions recorded:
Deployed Contracts
Currently you have no contract instances to interact with.

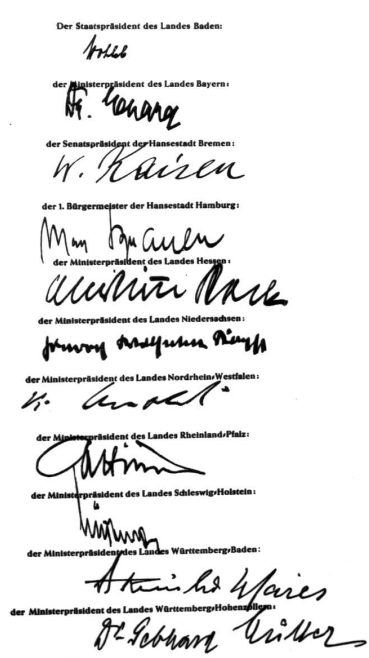

browser/wallet.sol
Compile
Run
Analysis
Testing
Debugger
Settings
browser
config
pragma solidity >=0.4.22 <0.6;
contract Wallet {
uint256 balance;
address payable owner;
constructor () public {
balance = 0;
owner = msg.sender;
}
function addfund() payable public retur
require (msg.sender == owner);
balance += msg.value;
return balance;
}
function withdraw() public {
require (msg.sender == owner);
selfdestruct(owner);
}
}
moz-extension://acaf48e3-a854...otification - Mozilla Firefox
Kovan Test Network
Test
New Contract
CONTRACT DEPLOYMENT
0
$0.00
DETAILS
DATA
EDIT
GAS FEE
0.000194
$0.03
AMOUNT + GAS FEE
TOTAL
0.000194
$0.03
Metamask
REJECT
CONFIRM
Environment
Injected Web3
Kovan (42)
Account
0x1c9...231a3 (2.99813089 ether)
Gas limit
3000000
Value
0
wei
Wallet
Deploy
or
At Address
Load contract from Address
Transactions recorded:
Deployed Contracts
Currently you have no contract instances to interact with.
web3.js
Remix IDE
[2] only remix transactions,
web3 version 1.0.0
ethers.js
swarmgw
compilers - contains currently loaded compi
Executing common command to interact with the
t these commands can also be included and run
Use exports/.register(key, obj)/.remove(key)/.
cutions.
creation of Wallet pending...
>

Beispiele

Beispiele

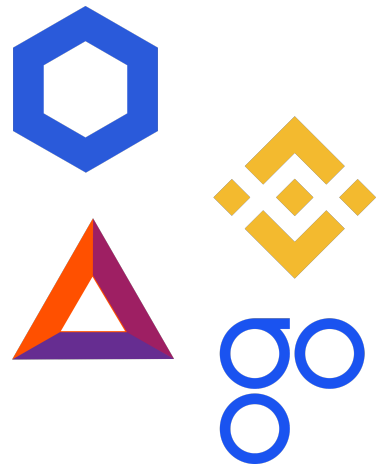
Beispiele

- Multi-Signatur-Wallets



Beispiele

- Multi-Signatur-Wallets
- ERC-20-Token



Beispiele

- Multi-Signatur-Wallets
- ERC-20-Token
- Handelsbörsen

Beispiele

- Multi-Signatur-Wallets
- ERC-20-Token
- Handelsbörsen
- Wetten

Beispiele

- Multi-Signatur-Wallets
- ERC-20-Token
- Handelsbörsen
- Wetten
- Cryptokitties

Komponenten einer Dapp

- Webapplikation mit JS (meist open source)
- Smart Contract(s)
- Blockchain

Softwarequalität

Entwicklungs-Workflow

Entwicklungs-Workflow

1. Contract in Solidity entwickeln

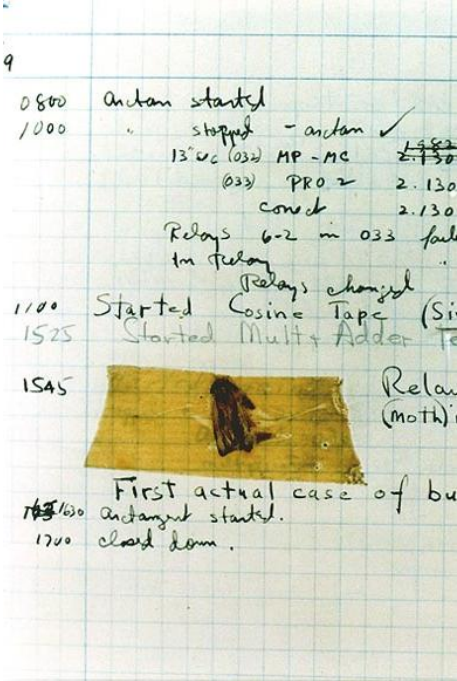
Entwicklungs-Workflow

1. Contract in Solidity entwickeln
2. mit Remix/Metamask ausrollen

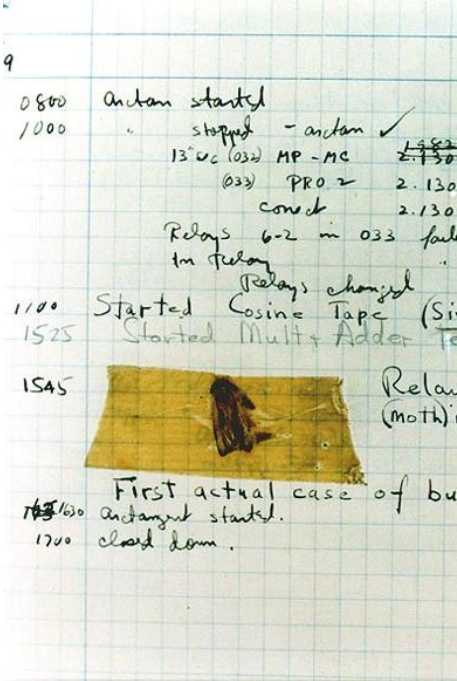
Entwicklungs-Workflow

1. Contract in Solidity entwickeln
2. mit Remix/Metamask ausrollen
3. ???

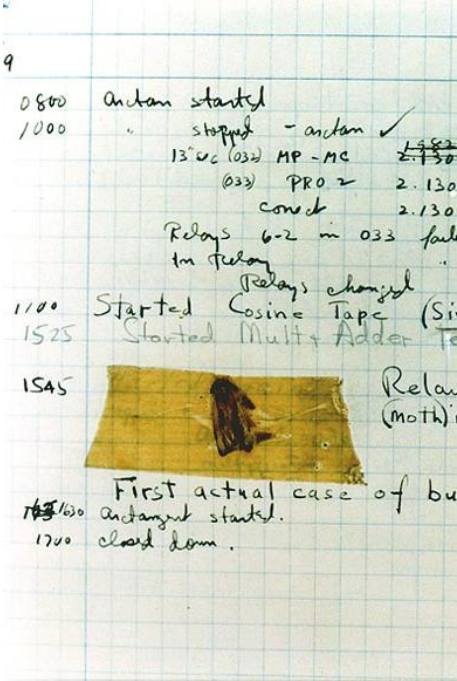
- Smart Contracts sind **unveränderlich**



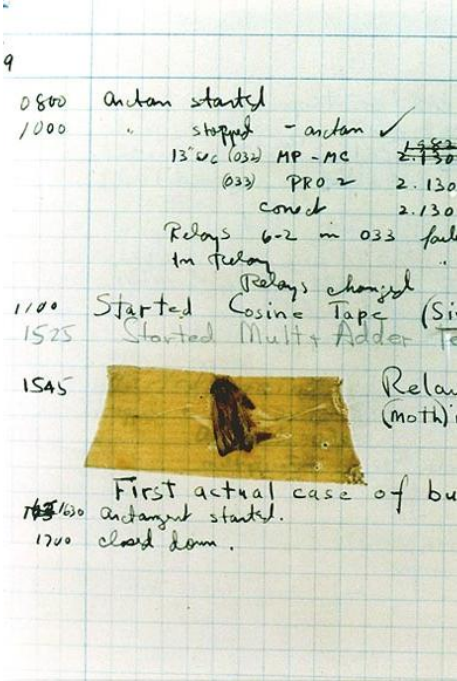
- Smart Contracts sind **unveränderlich**
- Transaktionen sind **endgültig**



- Smart Contracts sind **unveränderlich**
- Transaktionen sind **endgültig**
- es gibt **keine Kontrollinstanz***



- Smart Contracts sind **unveränderlich**
- Transaktionen sind **endgültig**
- es gibt **keine Kontrollinstanz***
- Fehler sind **nicht korrigierbar***



Fatale Fehler

The DAO (2016)

- *Decentralized Autonomous Organization*
- eine Art digitaler Geber von Wagniskapital
- Idee: Investor*innen stimmen über Kapitalnutzung ab
- Bug im Contract führte zum Abgreifen von 50 Millionen USD

Fatale Fehler

Parity Wallet (2017)

- Online Wallet für Ether und andere Token
- Bug im Contract führte zum Abgreifen von 30 Millionen USD

Fatale Fehler

Parity Wallet (2017)

- Online Wallet für Ether und andere Token
- Bug im Contract führte zum Abgreifen von 30 Millionen USD
- weiterer Bug führte zum Totalverlust von 360 Millionen USD

Qualitätssicherung

Truffle: Framework für den gesamten Entwicklungszyklus

- Testing (Solidity und JavaScript)
- Deployment
- Migrations

Code is Law

> *However, these more complex agreements, with greater engagement with real world goods and services, highlight the necessity of effective dispute resolution, as well as indicate a necessary interrelation with territorial legal systems.*

Code is Law

> *However, these more complex agreements, with greater engagement with real world goods and services, highlight the necessity of effective dispute resolution, as well as indicate a necessary interrelation with territorial legal systems.*
> *Contestation mechanisms are necessary to 'soften' the effects of self-executing 'smart contracts', and make transactions reversible, allowing the outcomes of dispute resolution to be enforced.*

Goldenfein & Later: "Legal Engineering on the Blockchain: 'Smart Contracts' as Legal Conduct" (2018)

Hürden für Smart Contracts

- bisher keine etablierten formalen Methoden
- unklare Jurisdiktion
- Nachvollziehbarkeit des Codes

Varianten von Ethereum

Ethereum Classic

- *Hard Fork* im Juli 2016
- Ursache: Unstimmigkeit über Vorgehen nach DAO-Hack
- Platz 18 in Marktkapitalisierung

Proof of Work

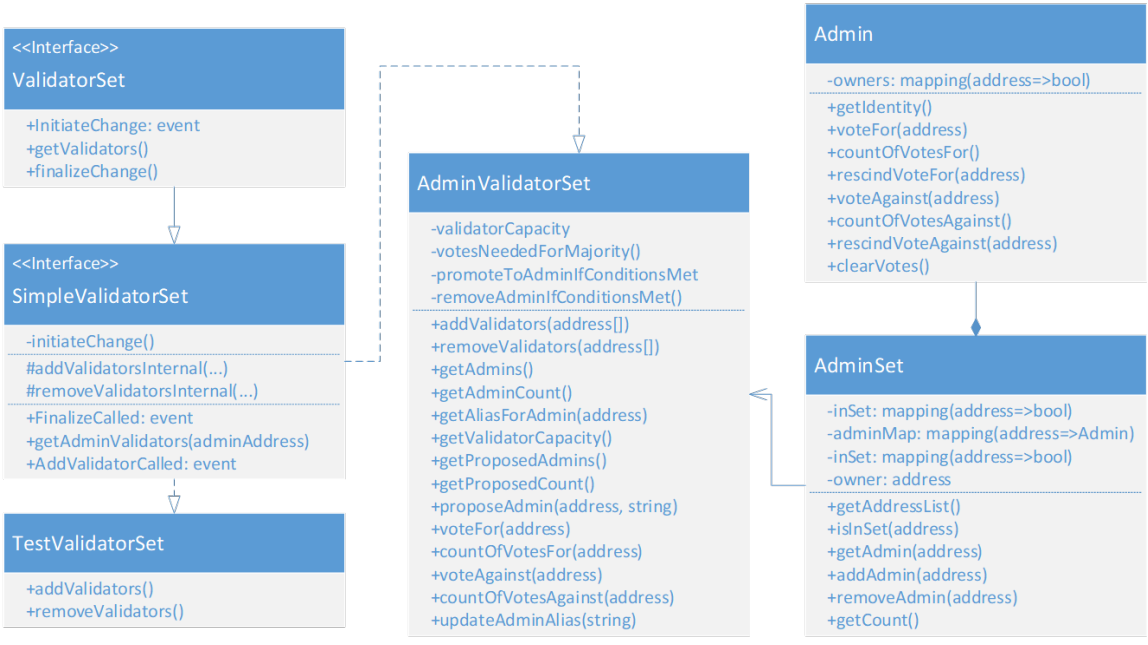
- eingeführt durch Bitcoin
- Rückgrat der meisten Kryptowährungen
- basiert auf Investition von Strom

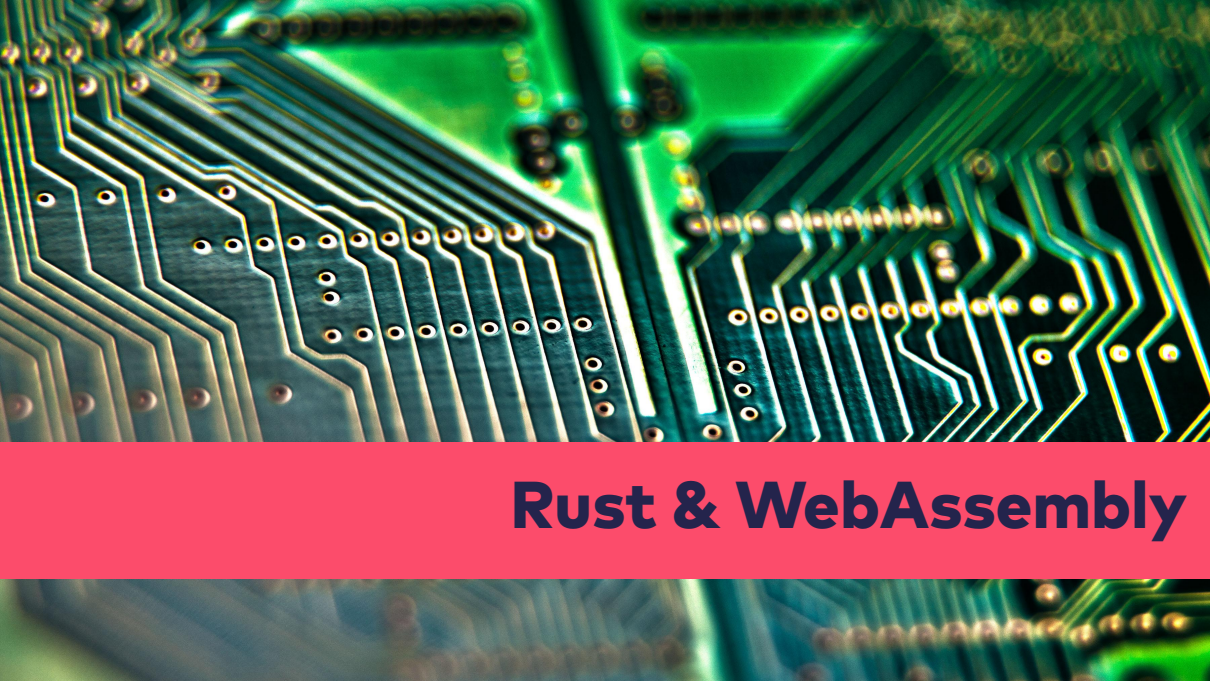
Proof of Stake

- experimentelle(s) Verfahren
- basiert auf Abstimmung über Blöcke
- Gewichtung nach *Stake*: z.B. Vermögen oder Vermögensalter
- als Ziel für Ethereum vorgesehen
- Umsetzung unklar

Proof of Authority

- ein Konsortium einigt sich über vertrauenswürdige Knoten
- abwechselnde Erzeugung von Blöcken
- keine Transaktionsgebühr nötig
- Konsortium kann selbst über Smart Contract gesteuert werden


<<Interface>>
ValidatorSet
+InitiateChange: event
+getValidators()
+finalizeChange()
<<Interface>>
SimpleValidatorSet
-initiateChange()
#addValidatorsInternal(...)
#removeValidatorsInternal(...)
+FinalizeCalled: event
+getAdminValidators(adminAddress)
+AddValidatorCalled: event
TestValidatorSet
+addValidators()
+removeValidators()
AdminValidatorSet
-validatorCapacity
-votesNeededForMajority()
-promoteToAdminIfConditionsMet
-removeAdminIfConditionsMet()
+addValidators(address[])
+removeValidators(address[])
+getAdmins()
+getAdminCount()
+getAliasForAdmin(address)
+getValidatorCapacity()
+getProposedAdmins()
+getProposedCount()
+proposeAdmin(address, string)
+voteFor(address)
+countOfVotesFor(address)
+voteAgainst(address)
+countOfVotesAgainst(address)
+updateAdminAlias(string)
Admin
-owners: mapping(address=>bool)
+getIdentity()
+voteFor(address)
+countOfVotesFor()
+rescindVoteFor(address)
+voteAgainst(address)
+countOfVotesAgainst()
+rescindVoteAgainst(address)
+clearVotes()
AdminSet
-inSet: mapping(address=>bool)
-adminMap: mapping(address=>Admin)
-inSet: mapping(address=>bool)
-owner: address
+getAddressList()
+isInSet(address)
+getAdmin(address)
+addAdmin(address)
+removeAdmin(address)
+getCount()



Rust & WebAssembly

Rust & WebAssembly

Rust

- maschinennahe Sprache von Mozilla
- Fokus auf Speichersicherheit
- kein GC

→

WebAssembly

- Assemblersprache für den Browser
- läuft in der JS-Sandbox
- kein Zugriff auf den DOM

EWASM

- WebAssembly auf der EVM: vielversprechende Entwicklung
- derzeit noch unspezifiziert
- experimentelle Implementierung in Parity

```rust
#[eth_abi(CAEndpoint, CAClient)]
pub trait CAInterface {
    fn constructor(&mut self);

    #[constant]
    fn owner(&mut self) -> U256;
}

pub struct CAContract;

impl CAInterface for CAContract {
    fn constructor(&mut self) {
        OWNER.write_as(pwasm_ethereum::sender());
    }

    fn owner(&mut self) -> U256 {
        OWNER.read_as()
    }
}
```

Fazit

Fazit

- Ethereum ist bisher größte Plattform für Dapps
- hohe Entwicklungsgeschwindigkeit
- leidet an mangelhaftem Sprachdesign
- Proof of Authority und EWASM vielversprechend, aber noch in den Kinderschuhen

Q & A

INNOQ
www.innoq.com

Lars Hupel

lars.hupel@innoq.com

@larsr_h

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim a. Rh.
Germany
+49 2173 3366-0

Ohlauer Str. 43
10999 Berlin
Germany

Ludwigstr. 180 E
63067 Offenbach
Germany

Kreuzstr. 16
80331 München
Germany

c/o WeWork
Hermannstrasse 13
20095 Hamburg
Germany

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
+41 41 743 01 11

Albulastr. 55
8048 Zürich
Switzerland

LARS HUPEL

Consultant
innoQ Deutschland GmbH

Lars enjoys programming in a variety of languages, including Scala, Haskell, and Rust. He is known as a frequent conference speaker and one of the founders of the Typelevel initiative which is dedicated to providing principled, type-driven Scala libraries.

Bildquellen

Decentralized Apps: https://medium.com/swlh/understanding-dapps-decentralized-applications-8f3668ebdc9a
Smart Contract: http://instaco.de/, https://www.ethereum.org/token
Akten: https://pixabay.com/photos/files-ddr-archive-1633406/
Ethereum Transactions: https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369
Platine: https://commons.wikimedia.org/wiki/File:Computer_Circuit_Board_MOD_45153619.jpg, Harland Quarrington, OGL 1.0
Gas Pump: https://commons.wikimedia.org/wiki/File:Antique_yard_vintage_gas_pumps,_Washington_LCCN2010630650.tif, Library of Congress
Treaty of Rome: https://en.wikipedia.org/wiki/File:Treaty_of_Rome.jpg, author unknown
Scared cat: https://pixabay.com/vectors/cat-scared-hunched-spooky-black-30788/
Coins: https://pixabay.com/photos/money-coins-euro-coins-currency-515058/
Macaroons: https://pixabay.com/photos/macaroon-dessert-sweet-bakery-food-2815075/
Power station: https://pixabay.com/photos/power-station-energy-electricity-374097/
UN General Assembly: https://www.flickr.com/photos/50371131@N04/29192350713, Gobierno de Chile, CC-BY
Bundesverwaltungsgericht: https://pixabay.com/photos/supreme-administrative-court-court-3687414/
Ratifikation des Grundgesetzes: https://de.wikipedia.org/w/index.php?title=Datei:Grundgesetz_Ratifikationsunterschriften.jpg&filetimestamp=20110526174813&
NYSE: https://pixabay.com/photos/stock-exchange-trading-floor-738671/
Poker chips: https://pixabay.com/photos/chips-play-poker-casino-gambling-2038348/
Contract UML: https://github.com/Azure-Samples/blockchain/blob/master/ledger/template/ethereum-on-azure/permissioning-contracts/validation-set/media/contract-uml.png
Server rack: https://pixabay.com/photos/server-cloud-development-business-1235959/
Lightning: https://pixabay.com/photos/lightning-storm-weather-sky-399853/
Ferris the Crab: https://www.rust-lang.org/what/wasm, MIT
Construction site: https://pixabay.com/photos/building-construction-site-cranes-768815/
Railroad: https://pixabay.com/photos/track-railway-line-rail-traffic-3670209/