RareSkills - Advanced Solidity

Meek Msaki (Meek#6464)

May 2023

Advanced solidity bootcamp with Dominik Teiml. Summary of the ethereum yellow paper.

1 Chapter 1

2 Chapter 2

3 Chapter 3

- $\boldsymbol{\sigma}$ - (sigma) World state
- $\boldsymbol{\mu}$ - (mu) Machine state
- $\boldsymbol{\Upsilon}$ - (upsilon) Ethereum state transition
- C - General cost function
- KEC - Keccak-256 hash function
- $KEC512$ - Keccak512 hash function
- T - Ethereum transaction
- δ - (delta) the number of items required on a stack for a given operation
- $\mathbf{o}$ - the output data of a message call
- $\mathbb{N}$ - a set of scalar non-negative numbers e.g. set of integers smaller than 2^{256} denoted by $\mathbb{N}_{256}$
- $\mathbb{B}$ - a set of byte sequences, e.g. bytes 32 is denoted by $\mathbb{B}_{32}$
- f - a function
- ℓ - a function which evaluates the last item in the given sequence

4 Chapter 4

4.1 The World State

- $\boldsymbol{\sigma}$ - represents world state or state. It is a mapping between addresses (160-bit) and account states, a data structure serialised as Recursive Length Prefix (RLP)
  - $\boldsymbol{\sigma}[a]$ = **Account state** - The state of ethereum accounts has four fields
    * $\boldsymbol{\sigma}[a]_n$ = account's **nounce state**
    * $\boldsymbol{\sigma}[a]_b$ = account's **balance state**
    * $\boldsymbol{\sigma}[a]_s$ = 256-bit hash **storageRoot** of merkle patricia trie root node that encodes storage contents (a mapping between 256-bit integer values)

· For Eternally Owned Accounts, $\boldsymbol{\sigma}[a]_s = \varnothing$
· For Contract accounts $\boldsymbol{\sigma}[a]_s \neq \varnothing$
· It's not hash of the merkle trie root but for the key/value pairs stored within $\boldsymbol{\sigma}[a]_s \equiv TRIE(L_I^*((\boldsymbol{\sigma}[a]_s))$

* $\boldsymbol{\sigma}[a]_c$ = **codeHash** - Keccak 256-bit hash of contract bytecode that gets executed when the account receives a message call
  · For Eternally Owned Accounts, $\boldsymbol{\sigma}[a]_c = \varnothing$
  · For contract accounts $\boldsymbol{\sigma}[a]_c \neq \varnothing$
  - $KEC(\mathbf{b}) = \boldsymbol{\sigma}[a]_c$, $\mathbf{b}$ to denote the contract's EVM bytecode

– An account is *empty* if it has no code $\boldsymbol{\sigma}[a]_c = \varnothing$, zero nonce $\boldsymbol{\sigma}[a]_n = 0$ and zero balance $\boldsymbol{\sigma}[a]_b = 0$
  * $EMPTY(\boldsymbol{\sigma}, a) \equiv \boldsymbol{\sigma}[a]_c = KEC(()) \wedge \boldsymbol{\sigma}[a]_n = 0 \wedge \boldsymbol{\sigma}[a]_b = 0$

– An account is *dead* if its account state is non-existent or empty
  * $DEAD(\boldsymbol{\sigma}, a) \equiv \boldsymbol{\sigma}[a] = \varnothing \vee EMPTY(\boldsymbol{\sigma}, a)$

4.2 The Transaction

- T - a single cryptographically sigend instruction by an EOA.
- S - maps transaction to the sender with ECDSA SECP-256k1 curve (hash of the transaction excepting three signature fields).
- Assert the sender of a transaction T represents $S(T)$
- $L_T(T) \equiv \begin{cases} (T_n, T_p, T_g, T_t, T_v, T_i, T_w, T_r, T_s) \text{ if } T_t = \varnothing \\ (T_n, T_p, T_g, T_t, T_v, T_d, T_w, T_r, T_s) \textit{ otherwise} \end{cases}$ - Specifies how to serialize a transaction

  The sender of a transaction cannot be a contract

  There are two subtype of transactions

  – Those that result in message calls
  – Those that result in creation of new accounts with associated code ('contract creation')

  EIP-1559: Fee market change (type 2) transactions have:

  – T_n - Transaction nounce for the sender, $T_n \in \mathbb{N}_{256}$
  – T_g - Gas Price, $T_p \in \mathbb{N}_{256}$
  – T_g - Gas Limit, $T_g \in \mathbb{N}_{256}$
  – T_t - To (160-bit address), $T_t \in \begin{cases} \mathbb{B}_{20} \text{ if } T_t \neq \varnothing \\ \mathbb{B}_0 \text{ contract creation} \end{cases}$
  – T_i - EVM-code for account initialization, $T_i \in \mathbb{B}$
  – T_r, T_s - Signature of the transaction used to determine the sender of the transaction
  – T_w - **ChainId** and **yParity** are combined to form single value $T_w = 2\beta + 35 + T_y$ (see EIP-155 by Buterin [2016b]), $T_w \in \mathbb{N}_{256}$
  – T_x - EIP-2718 transaction type, $T_x \in \{1, 2\}$
  – T_p - Gas Price, $T_p \in \mathbb{N}_{256}$
  – T_g - Gas Limit, $T_g \in \mathbb{N}_{256}$
  – T_v - Value, $T_v \in \mathbb{N}_{256}$
  – T_r, T_s - Signature of the transaction used to determine the sender of the transaction, $T_r \in \mathbb{N}_{256}$, $T_s \in \mathbb{N}_{256}$
  – $T_{\boldsymbol{A}}$ - Access list (Legacy transactions don't have access list)
  – T_c - Chain ID, $T_c = \beta$

- T_y - yParity signature, $T_y \in \mathbb{N}_1$ (bool)
- T_i - init is an unlimited size byte array specifying the EVM-code for contract initialization procedure, it is fragment that returns a **body**, $T_i \in \mathbb{B}$
- T_d - data byte array specifying input data of a message call, $T_d \in \mathbb{B}$

```
@dataclass
class Transaction1559Payload:
        chain_id: int = 0
        signer_nonce: int = 0
        max_priority_fee_per_gas: int = 0
        max_fee_per_gas: int = 0
        gas_limit: int = 0
        destination: int = 0
        amount: int = 0
        payload: bytes = bytes()
        access_list: List[Tuple[int, List[int]]] = field(default_factory=list)
        signature_y_parity: bool = False
        signature_r: int = 0
        signature_s: int = 0
```

EIP-2930 (type 1) transactions also have:

- $T_{\boldsymbol{A}}$ - Access list (Legacy transactions don't have access list)
- T_c - Chain ID must equal chain ID β, $T_c = \beta$
- T_y - yParity signature

```
@dataclass
class Transaction2930Payload:
        chain_id: int = 0
        signer_nonce: int = 0
        gas_price: int = 0
        gas_limit: int = 0
        destination: int = 0
        amount: int = 0
        payload: bytes = bytes()
        access_list: List[Tuple[int, List[int]]] = field(default_factory=list)
        signature_y_parity: bool = False
        signature_r: int = 0
        signature_s: int = 0
```

4.3 The Block

The Block, $B \equiv (B_H, B_T, B_U)$, is a collection of the following relevant pieces of information:

1. **T** - information corresponding to transactions
   - B_T - A series of transactions from this block
2. **U** - A set of block headers, ($ommers^2$) or uncles
   - B_U - A list of ommer block headers
3. H - represents block *header*, which contains: $\mathbb{B} = \{B : B \in \mathbb{B} \vee \|B\| = n\}$
   (a) H_p - **parentHash** - keccak256-bit hash of the parent's block header
      - $H_p \in \mathbb{B}_{32}$
      - $H_p \equiv KEC(P(B_H))$ where $P(B_H)$ is the parent block header of B
      - $TRIE(L_s(\sigma)) = P(B_H)_{H_r}$

(b) H_o - **ommersHash** - keccak256-bit hash of the ommers list of this block
   - $H_o \in \mathbb{B}_{32}$
   - $H_o \equiv KEC(RLP(L_H^*(B_U)))$

(c) H_c - **beneficiary** - 160-bit address that receives of all fees collected from successful mining this block
   - $H_c \in \mathbb{B}_{20}$

(d) H_r - **stateRoot** - keccak256 hash of the root node of the state trie after all transaction are executed on this block, $H_r \equiv KEC(TRIE(L_s(\Pi(\sigma, B))))$
   - $H_r \in \mathbb{B}_{32}$
   - Π - transaction state's accummulation function

(e) H_t - **transactionRoot** - keccak26-Bit hash of trie's root node with this block's transactions
   - $H_t \in \mathbb{B}_{32}$
   - $H_t \equiv KEC(TRIE(\{\forall i < \|B_{\mathbf{T}}\|, i \in \mathbb{N} : p_T(i, B_{\mathbf{T}}[i])\}))$

(f) H_e - **receiptsRoot** - keccak256-bit hash of the trie's root node containing receipt's of each transaction from this block, $H_e \equiv KEC(TRIE(\{\forall i < \|B_R\|, i \in \mathbb{N} : p_R(i, B_R[i])\}))$
   - $H_e \in \mathbb{B}_{32}$
   - B_R - values steming from the computation of transactions, specificaly transaction receipts

(g) H_b - **logsBloom** - Bloom filter from indexable info (logger addres and log topics) contained in each log entry from the receipt of each transaction in this block
   - $H_b \in \mathbb{B}_{256}$

(h) H_d - **difficulty** - a scalar value of the difficulty level of this block
   - $H_d \in \mathbb{N}$

(i) H_i - **number** - a scalar value equal to the number of ancestor blocks
   - $H_i \in \mathbb{N}$

(j) H_l - **gasLimit** - a scalar value equal to the current limit of gas expenditure per block
   - $H_l \in \mathbb{N}$

(k) H_g - **gasUsed** - a scalar value equal to the total gas used in transactions in this block
   - $H_g \in \mathbb{N}$

(l) H_s - **timestamp** - a scalar value equal to the output of Unix's time() at the block's inception
   - $H_s \in \mathbb{N}_{256}$

(m) H_x - **extraData** - arbitrary byte array data relevant to this block, must be 32 bytes or less
   - $H_x \in \mathbb{B}$

(n) H_m - **mixHash** - 256-bit hash which combined with nonce, proves a sufficient amount of computation has been carried out in this block
   - $H_m \in \mathbb{B}_{32}$

(o) H_n - **nonce** - 64-but value which combined with mix-hash proves sufficient computation has been carried out
   - $H_n \in \mathbb{B}_8$

4.3.1 Transaction Receipt

- $B_R[i]$ - receipt for the i^{th} transaction in an index-keyed trie who's root is recorded in the header as H_e
- $R \equiv (R_x, R_z, R_u, R_b, R_l)$
- $L_R(R) \equiv (R_z, R_u, R_b, R_l)$ - the L_R function prepares tx receipt to be transformed into an RLP-serialized bytes array

- R - Transaction receipt (a tuple of 5 items):
  1. R_x - equal to the corresponding transaction type
  2. R_z - status code of the transaction, $R_z \in \mathbb{N}$ (non-negative integer)
  3. R_u - cumulative gas used, $R_u \in \mathbb{N}$ (non-negative integer)
  4. R_l - a series of logs entries $(O_0, O_1, ...)$ created through execution of the transaction:
     (a) A log entry, O, is a tuple of: $O \equiv (O_a, (O_{t0}, O_{t1}, ...), O_d)$
        i. O_a - logger's address, $O_a \in \mathbb{B}_{20}$
        ii. O_t - series of 32-byte log topics, $\forall x \in O_t : x \in \mathbb{B}_{32}$
        iii. O_d - some number of bytes data, $O_d \in \mathbb{B}$
  5. R_b - the Bloom filter composed from the information in the logs, $R_b \in \mathbb{B}_{256}$
     (a) M - a function (bloom filter) to reduce a log entry into a single 256-byte hash:
        i. $M(O) \equiv \bigvee_{x \in O_a \cup O_t}(M_{3:2048}(x))$
           A. $M_{3:2048}$ - a specialized Bloom filter that sets three bits out of 2048, given an arbitrary byte sequence
           B. $M_{3:2048}$ - takes the low-order 11 bits of each of the first 3 pairs of bytes in a keccak-256 hash of the byte sequence
           C. $M_{3:2048}(\mathbf{x} : \mathbf{x} \in \mathbb{B}) \equiv \mathbf{y} : \mathbf{y} \in \mathbb{B}_{256}$ where:
              - $\mathbf{y} = (0, 0, 0, ..., 0)$ except:
              - $\forall i \in \{0, 2, 4\} : B_{2047-m(\mathbf{x},i)}(\mathbf{y}) = 1$
              - B - a bit reference function
           D. $m(\mathbf{x}, i) \equiv KEC(\mathbf{x})[i, i+1] \mod 2048$

4.3.2 Holistic Validity

- identity of state when transactions executed in order on the base state
  - $H_r \equiv TRIE(L_s(\Pi(\sigma, B)))\wedge$
  - $H_o \equiv KEC(RLP(L_H^*(B_U)))\wedge$
  - $H_t \equiv TRIE(\{\forall i < \|B_\mathbf{T}\|, i \in \mathbb{N} : p_T(i, B_\mathbf{T}[i])\})\wedge$
  - $H_e \equiv TRIE(\{\forall i < \|B_R\|, i \in \mathbb{N} : p_R(i, B_R[i])\})\wedge$
  - $\bigvee_{r \in B_R}(\mathbf{r}_b)$

(TODO: to be continued... pg 6/41)

4.3.3 Serialization

- $L_H(H)$ - preparation function for block header
  - $L_H(H) \equiv (H_p, H_o, H_r, H_t, H_e, H_b, H_d, H_i, H_l, H_g, H_s, H_x, H_m, H_n)$
- $L_B(B)$ - preparation function for block
  - $L_B(B) \equiv (L_H(B_H), \tilde{L}_T^*(B_H), L_H^*(B_U))$
  - $\tilde{L}_T$ - takes special care of EIP-2718 transactions

4.3.4 Block Header Validity

- $P(H) \equiv B' : KEC(RPL(B'_H)) = H_p$
- $H_i \equiv P(H)_{Hi} + 1$ - the block number is the parent's block number uncremented by one
- $D(H)$ - canonical difficulty fo a block header H
- ς_2 - *Homestead* diffuculty parameter
- ϵ - exponetial difficulty symbol

5 Chapter 5

6 Chapter 6 (Transaction Execution)

- Υ - state transition function
- $\boldsymbol{\sigma}'$ - the post-transactional state
- $\boldsymbol{\sigma}' = \Upsilon(\sigma, T)$ - Υ is the function, T is the transaction and σ the state
- Υ^g - evaluates amount of gas used in the transaction
- Υ^l - evaluates transaction's accrued log items
- Υ^z - evaluates the status code resulting from the transaction

6.1 Substate

Accrued substate (A) is the information that is acted upon immediately following the transaction execution

- $A \equiv (A_s, A_l, A_t, A_r, A_a, A_K)$ - A is a tuple
- A_s - the self destruct set, a set of accounts that will be discarded following transaction completion
- A_l - log series of archived and indexable 'checkpoints' in VM code execution
- A_t - set of touched accounts, which the empty ones are deleted at the end of transaction
- A_r - the refund balance, from SSTORE instruction when contract storage is reset to zero from non-zero
- A_a - the set of accessed account addresses
- A_K - a tuple of a 20-byte account address and a 32-byte storage slot (a set of storage keys)
- A^0 - an empty accrued substate
- $A^0 \equiv (\varnothing, (), \varnothing, 0, \pi, \varnothing)$, where: π is a set of precompiled addresses
- π - a set of all precompiled addresses

6.2 Execution

- g_0 - amount of gas this transaction requires to be paid prior to execution, intrinsic gas g_0

$$g_0 \equiv \sum_{i \in T_i, T_d} \begin{cases} G_{txdatazero} \text{ if } i = 0 \\ G_{txdatanonzero} \text{ otherwise} \end{cases} + \begin{cases} G_{txzero} \text{ if } T_t = \varnothing \\ 0 \text{ otherwise} \end{cases}$$

$$+ G_{transaction} + \sum_{j=0}^{\|T_A\|-1} \left(G_{accesslistaddress} + \|T_A[j]_s\| G_{accessliststorage}\right)$$

  - T_i, T_d is the series of bytes of the transaction's associated data and initialisation EVM-code
  - $G_{txcreate}$ iis addres if the transaction is creating a contract
  - $G_{accesslistaddress}$ and $G_{accessliststorage}$ are the cost of warming up account and storage access
  - G (defined in appendix G) - is a tuple of scalar values corresponding to relative costs in gas of operations

- C - the general **gas cost** function
-

Name	Value	Description
G_{zero}	0	0 gas operations of the set W_{zero} = {STOP, RETURN, REVERT}
$G_{jumpdest}$	1	gas paid for JUMPDEST operation
G_{base}	2	gas paid for operations of set W_{base} = {ADDRESS, ORIGIN, CALLER, CALLVALUE, CALLDATASIZE, CODESIZE, COINBASE, TIMESTAMP, NUMBER, DIFFICULTY, GASLIMIT, CAHINID, RETURNDATASIZE, POP, POP, MSIZE, Gas}
$G_{verylow}$	3	gas paid for operations of set $W_{verylow}$ = {ADD, SUB, NOT, LT, GT, SLT, SGT, EQ, ISZERO, AND, OR, XOR, BYTE, SHL, SHR, SAR, CALLDATALOAD, MLOAD, MSTORE, MSTORE8, PUSH*, DUP*, SWAP*}
G_{low}	5	gas paid for operations of set W_{low} = {MUL, DIV, SDIV, MOD, SMOD, SIGNEXTEND, SELFBALANCE}
G_{mid}	8	gas paid for operations of set W_{mid} = {ADDMOD, MULMOD, JUMP}
G_{high}	10	gas paid for operations of set W_{high} = {JUMPI}
$G_{warmaccess}$	100	cost for a warm account or storage
$G_{accesslistaddress}$	2400	cost for warming up an account with the acess list
$G_{accessliststorage}$	1900	cost for warming up a storage with the access list
$G_{coldaccountaccess}$	2600	cost for a cold account access
$G_{coldload}$	2100	cost for a cold account access
G_{sset}	20000	cost for an SSTORE operation from non-zero to zero
G_{sreset}	2900	cost for an SSTORE operation from zero to zero (unchanged)
R_{sclear}	15000	refund when SSTORE operation on a storage is set from non-zero to zero
$R_{selfdestruct}$	24000	refund for self-destructing and account
$G_{selfdescturct}$	5000	gas paid for a SELFDESTRUCT operation
G_{create}	32000	gas paid for a CREATE operation
$G_{codedeposit}$	200	cost paid per byte for a CREATE operation to success in placing code to state
$G_{callvalue}$	9000	gas paid for a non-zero value transfer as part of the CALL operation
$G_{callstipend}$	2300	a stipend for the called contract subtracted from the $G_{callvalue}$ for a non-zero value
$G_{newaccount}$	25000	gas paid for a CALL or SELFDESTRUCT operations which creates a new account
G_{exp}	10	partial payment for an EXP operations
G_{expbye}	50	partial payment when multiplied by the number of bytes in the exponent for EXP operation
G_{memory}	3	gas paid for every additional word when expanding memory
$G_{txcreate}$	32000	gas paid by all contract-creating transactions after *Honstead* transition
$G_{txdatazero}$	4	gas paid for every zero bytes of data or code for a transaction
$G_{txdatanonzero}$	16	gas paid for every non-zero bytes of data or code for a transaction
$G_{transaction}$	21000	gas paid for every transaction
G_{log}	375	partial payment for a LOG operation
$G_{logdata}$	8	gas paid for each bytes in a LOG operation's data
$G_{logtopic}$	375	gas paid for each topic of a LOF operation
$G_{keccak256}$	30	gas paid for each KECCAK256 operation
$G_{keccak256word}$	60	gas paid for each word (rounded up) for input data to a KECCAK256 operation
G_{copy}	3	partial payment for set W_{copy} = {CALLDATACOPY, CODECOPY, RETURNDATACOPY} operations, multiplied by words copied, rounded up
$G_{blockhash}$	20	gas pid for each BLOCKHASH operation

–

–

–