Ethereum Transactions

Saravanan Vijayakumaran
sarva@ee.iitb.ac.in

Department of Electrical Engineering
Indian Institute of Technology Bombay

August 29, 2019

World State and Transactions

- World state consists of a trie storing key/value pairs
  - For accounts, key is 20-byte account address
  - Account value is [nonce, balance, storageRoot, codeHash]
- Transactions cause state transitions
- $\boldsymbol{\sigma}_t$ = Current state, $\boldsymbol{\sigma}_{t+1}$ = Next state, T = Transaction

$$\boldsymbol{\sigma}_{t+1} = \Upsilon(\boldsymbol{\sigma}_t, T)$$

- Transactions are included in the blocks
- Given genesis block state and blockchain, current state can be reconstructed

Ethereum Transaction Format

`nonce`	≤ 32 bytes
`gasprice`	≤ 32 bytes
`startgas`	≤ 32 bytes
`to`	1 or 20 bytes
`value`	≤ 32 bytes
`init/data`	≥ 0 bytes
`v`	≥ 1 bytes
`r`	32 bytes
`s`	32 bytes

- Ethereum transactions are of two types
  - Contract creation
  - Message calls
- Contract creation transactions have EVM code in `init` field
  - Execution of `init` code returns a `body` which will be installed
- Message calls specify a function and its inputs in `data` field
- Transfer of ether between EOAs is considered a message call
  - Sender can insert arbitrary info in `data` field

nonce

`nonce`	≤ 32 bytes
`gasprice`	≤ 32 bytes
`startgas`	≤ 32 bytes
`to`	1 or 20 bytes
`value`	≤ 32 bytes
`init/data`	≥ 0 bytes
`v`	≥ 1 bytes
`r`	32 bytes
`s`	32 bytes

- Number of transactions sent by the sender address
- Prevents transaction replay
- First transaction has `nonce` equal to 0

gasprice and startgas

`nonce`	≤ 32 bytes
`gasprice`	≤ 32 bytes
`startgas`	≤ 32 bytes
`to`	1 or 20 bytes
`value`	≤ 32 bytes
`init/data`	≥ 0 bytes
`v`	≥ 1 bytes
`r`	32 bytes
`s`	32 bytes

- Each operation in a transaction execution costs some *gas*
- gasprice = Number of Wei to be paid per unit of gas used during transaction execution
- startgas = Maximum gas that can be consumed during transaction execution
  - gasprice*startgas Wei are deducted from sender's account
  - Any unused gas is refunded to sender's account at same rate
- Any unrefunded Ether goes to miner

Fee Schedule

- A tuple of 31 values which define gas costs of operations
- Partial fee schedule (full schedule in Appendix G of yellow paper)

Name	Value	Description
G_{base}	2	Paid for operations in set W_{base}.
$G_{verylow}$	3	Paid for operations in set $W_{verylow}$.
G_{low}	5	Paid for operations in set W_{low}.
G_{mid}	8	Paid for operations in set W_{mid}.
G_{high}	10	Paid for operations in set W_{high}.
G_{call}	700	Paid for a CALL operation.
$G_{transaction}$	21000	Paid for every transaction.
$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	68	Paid for every non-zero byte of data or code for a transaction.
G_{txcreate}	32000	Paid by all contract-creating transactions
$G_{codedeposit}$	200	Paid per byte for a CREATE operation
$G_{selfdestruct}$	5000	Amount of gas to pay for a SELFDESTRUCT operation.
$R_{selfdestruct}$	24000	Refund given for self-destructing an account.
G_{sha3}	30	Paid for each SHA3 operation.

to and value

`nonce`	≤ 32	bytes
`gasprice`	≤ 32	bytes
`startgas`	≤ 32	bytes
`to`	1 or 20	bytes
`value`	≤ 32	bytes
`init/data`	≥ 0	bytes
`v`	≥ 1	bytes
`r`	32	bytes
`s`	32	bytes

- For contraction creation transaction, `to` is empty
  - RLP encodes empty byte array as `0x80`
  - Contract address = Right-most 20 bytes of Keccak-256 hash of `RLP`([senderAddress, nonce])
- For message calls, `to` contains the 20-byte address of recipient
- `value` is the number of Wei being transferred to recipient
  - In message calls, the receiving contract should have `payable` functions

Recursive Length Prefix Encoding

Recursive Length Prefix Encoding (1/3)

- Applications may need to store complex data structures
- RLP encoding is a method for serialization of such data
- Value to be serialized is either a byte array or a list of values
  - Examples: “abc”, [“abc”, [“def”, “ghi”], [“”]]

$$\mathtt{RLP}(\mathbf{x}) = \begin{cases} R_b(\mathbf{x}) & \text{if } \mathbf{x} \text{ is a byte array} \\ R_l(\mathbf{x}) & \text{otherwise} \end{cases}$$

- $\mathtt{BE}$ stands for big-endian representation of a positive integer

$$\mathtt{BE}(x) = (b_0, b_1, ...) : b_0 \neq 0 \wedge x = \sum_{n=0}^{n<\|\mathbf{b}\|} b_n \cdot 256^{\|\mathbf{b}\|-1-n}$$

Recursive Length Prefix Encoding (2/3)

- Byte array encoding

$$R_b(\mathbf{x}) = \begin{cases} \mathbf{x} & \text{if} \quad \|\mathbf{x}\| = 1 \wedge \mathbf{x}[0] < 128 \\ (128 + \|\mathbf{x}\|) \cdot \mathbf{x} & \text{else if} \quad \|\mathbf{x}\| < 56 \\ \big(183 + \big\|\mathtt{BE}(\|\mathbf{x}\|)\big\|\big) \cdot \mathtt{BE}(\|\mathbf{x}\|) \cdot \mathbf{x} & \text{else if} \ \big\|\mathtt{BE}(\|\mathbf{x}\|)\big\| \leq 8 \end{cases}$$

- $(a) \cdot (b) \cdot c = (a, b, c)$
- Examples
  - Encoding of 0xaabbcc = 0x83aabbcc
  - Encoding of empty byte array = 0x80
  - Encoding of 0x80 = 0x8180
  - Encoding of “Lorem ipsum dolor sit amet, consectetur adipisicing elit” = 0xb8, 0x38, ’L’, ’o’, ’r’, ’e’, ’m’, ’ ’, . . . , ’e’, ’l’, ’i’, ’t’
- Length of byte array is assumed to be less than 256^8
- First byte can be at most 191

Recursive Length Prefix Encoding (3/3)

- List encoding of $\mathbf{x} = [\mathbf{x}_0, \mathbf{x}_1, \ldots]$

$$
\begin{aligned}
R_l(\mathbf{x}) &= \begin{cases} (192 + \|s(\mathbf{x})\|) \cdot s(\mathbf{x}) & \text{if} \quad \|s(\mathbf{x})\| < 56 \\ \left(247 + \left\|\mathtt{BE}(\|s(\mathbf{x})\|)\right\|\right) \cdot \mathtt{BE}(\|s(\mathbf{x})\|) \cdot s(\mathbf{x}) & \text{otherwise} \end{cases} \\
s(\mathbf{x}) &= \mathtt{RLP}(\mathbf{x}_0) \cdot \mathtt{RLP}(\mathbf{x}_1) \ldots
\end{aligned}
$$

- Examples
  - Encoding of empty list [ ] = 0xc0
  - Encoding of list containing empty list [[ ]] = 0xc1 0xc0
  - Encoding of [[ ], [[ ]], [[ ], [[ ]]]] = 0xc7, 0xc0, 0xc1, 0xc0, 0xc3, 0xc0, 0xc1, 0xc0
- First byte of RLP encoded data specifies its type
  - 0x00, . . . , 0x7f $\implies$ byte
  - 0x80, . . . , 0xbf $\implies$ byte array
  - 0xc0, . . . , 0xff $\implies$ list

Reference: `https://github.com/ethereum/wiki/wiki/RLP`

Recovering Sender Address from a Transaction

v,r,s

`nonce`	≤ 32	bytes
`gasprice`	≤ 32	bytes
`startgas`	≤ 32	bytes
`to`	1 or 20	bytes
`value`	≤ 32	bytes
`init/data`	≥ 0	bytes
`v`	≥ 1	bytes
`r`	32	bytes
`s`	32	bytes

- (`r`, `s`) is the ECDSA signature on hash of remaining Tx fields
- Note that the sender's address is not a header field
- `v` enables recovery of sender's public key

secp256k1 Revisited

- Ethereum uses the same curve as Bitcoin for signatures
- $y^2 = x^3 + 7$ over $\mathbb{F}_p$ where

$$p = \underbrace{\texttt{FFFFFFFF} \ \cdots \ \texttt{FFFFFFFF}}_{\text{48 hexadecimal digits}} \ \texttt{FFFFFFFE} \ \texttt{FFFFFC2F}$$
$$= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

- $E \cup \mathcal{O}$ has cardinality n where

$$n = \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE}$$
$$\texttt{BAAEDCE6 AF48A03B BFD25E8C D0364141}$$

- Private key is $k \in \{1, 2, \ldots, n-1\}$
- Public key is kP where P is the base point of secp256k1
- Note that $p \approx 2^{256}$ and $n > 2^{256} - 2^{129}$

Public Key Recovery in ECDSA

- **Signer:** Has private key k and message m
  1. Compute $e = H(m)$
  2. Choose a random integer j from $\mathbb{Z}_n^*$
  3. Compute $jP = (x, y)$
  4. Calculate $r = x \bmod n$. If $r = 0$, go to step 2.
  5. Calculate $s = j^{-1}(e + kr) \bmod n$. If $s = 0$, go to step 2.
  6. Output (r, s) as signature for m
- **Verifier:** Has public key kP, message m, and signature (r, s)
  1. Calculate $e = H(m)$
  2. Calculate $j_1 = es^{-1} \bmod n$ and $j_2 = rs^{-1} \bmod n$
  3. Calculate the point $Q = j_1 P + j_2(kP)$
  4. If $Q = \mathcal{O}$, then the signature is invalid.
  5. If $Q \neq \mathcal{O}$, then let $Q = (x, y) \in \mathbb{F}_p^2$. Calculate $t = x \bmod n$. If $t = r$, the signature is valid.
- If $Q = (x, y)$ was available, then

$$kP = j_2^{-1}\left(Q - j_1 P\right)$$

- But we only have $r = x \bmod n$ where $x \in \mathbb{F}_p$

Recovery ID

- Since $p < 2^{256}$ and $n > 2^{256} - 2^{129}$, four possible choices for (x, y) given r
- Recall that (x, y) on the curve implies $(x, -y)$ on the curve
- Recovery ID encodes the four possibilities

Rec ID	x	y
0	r	even
1	r	odd
2	$r + n$	even
3	$r + n$	odd

- For historical reasons, recovery id is in range 27, 28, 29, 30
- Prior to Spurious Dragon hard fork at block 2,675,000 `v` was either 27 or 28
  - Chances of 29 or 30 is less than 1 in 2^{127}
  - `v` was not included in transaction hash for signature generation

Chain ID

- In EIP 155, transaction replay attack protection was proposed
- Chain IDs were defined for various networks

CHAIN_ID	Chain
1	Ethereum mainnet
3	Ropsten
61	Ethereum Classic mainnet
62	Ethereum Classic testnet

- After block 2,675,000, Tx field v equals 2 $\times$ CHAIN_ID + 35 or 2 $\times$ CHAIN_ID + 36
- Transaction hash for signature generation included CHAIN_ID
- Transactions with v equal to 27 to 28 still valid but insecure against replay attack

References

- Yellow paper `https://ethereum.github.io/yellowpaper/paper.pdf`
- Pyethereum `https://github.com/ethereum/pyethereum`
- Pyrlp `https://github.com/ethereum/pyrlp`
- Spurious Dragon hard fork `https://blog.ethereum.org/2016/11/18/hard-fork-no-4-spurious-dragon/`
- EIP 155: Simple replay attack protection `https://github.com/ethereum/EIPs/blob/master/EIPS/eip-155.md`