Ethereum Transactions

Saravanan Vijayakumaran

Department of Electrical Engineering
Indian Institute of Technology Bombay

February 15, 2024

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
- A transaction can only be initiated by an EOA, not a contract
- Ethereum transactions are of two types
  - Contract creation
  - Message calls (ETH transfers or contract method invocations)
- A message call transaction can result in further message calls
- As of the London upgrade (block 12965000), there are three types of transactions in Ethereum

Legacy Transaction Format

Legacy Transaction Format

- Type 0 or legacy transaction
  - `rlp([nonce, gasPrice, gasLimit, to, value, data, v, r, s])`
- `nonce`
  - Number of transactions sent by the sender address
  - Prevents transaction replay
  - First transaction has `nonce` equal to 0
- `gasPrice, gasLimit`
  - Each operation in a transaction execution costs some *gas*
  - `gasprice` = Number of Wei to be paid per unit of gas used during transaction execution
  - `gasLimit` = Maximum gas that can be consumed during transaction execution
  - `gasprice` × `gasLimit` 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

Legacy Transaction Format

- Type 0 or legacy transaction
  - `rlp([nonce, gasPrice, gasLimit, to, value, data, v, r, s])`
- `to`
  - 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`
  - The number of Wei being transferred to recipient
  - In message calls, the receiving contract should have `payable` functions

Legacy Transaction Format

- Type 0 or legacy transaction
  - `rlp([nonce, gasPrice, gasLimit, to, value, data, v, r, s])`
- `init/data`
  - This field is also called **calldata**
  - 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
    - The first 4 bytes of the data field specify the function
    - The remaining bytes specify the inputs to the function
    - The first 4 bytes of the Keccak hash of the function signature is used
  - Transfer of ether between EOAs is considered a message call
    - Sender can insert arbitrary info in `data` field
- `v, r, s`
  - (`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
4	Rinkeby
61	Ethereum Classic mainnet
62	Ethereum Classic testnet

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

Blockchain Forks

- Temporary Forks
  - When two miners mine a block at almost the same time
- Soft forks and hard forks
  - Caused by changes to the consensus rules
  - Consensus rules = Rules determining validity of blocks and transactions
- Soft forks
  - Backward compatible rule changes
  - Nodes which do not upgrade still consider blocks produced under new rules valid
  - **Example:** Block size limit reduced to 500 KB from 1 MB
    - Sub-500 KB blocks produced by upgraded miners will be considered valid by non-upgraded nodes
    - Blocks with size larger than 500 KB produced by non-upgraded miners will be rejected by upgraded nodes
  - Soft fork success requires nodes controlling a majority of the hashpower to upgrade to new rules
- Hard forks
  - Not backward compatible rule changes
  - Hard fork success requires all nodes to upgrade

Type 1 Transaction Format

Type 1 Transaction Format

- Type 1 transaction
  - `0x01 || rlp([chainId, nonce, gasPrice, gasLimit, to, value, data, accessList, signatureYParity, signatureR, signatureS])`
- Proposed in EIP 2930; included in Berlin hard fork (Apr 2021)
- New fields
  - `chainId`
  - `signatureYParity`
  - `accessList`
- The chain ID and *y*-coordinate parity fields were unbundled for simplicity
- The `accessList` is a list of contract addresses and storage slots
  - Each address costs 2400 gas and each storage slot costs 1900 gas
  - Subsequent accesses cost 100 gas each (warm access cost)
- Motivation
  - SLOAD cost was increased from 800 to 2100 gas in Berlin
  - Some contracts which assumed the lower gas cost broke

Access List

- Has form `[[20 bytes, [32 bytes...]]...]`
- Example

```
[
  [
      "0xde0b295669a9fd93d5f28d9ec85e40f4cb697bae",
      [
          "0x0000000000000..00000000000000000000000000003",
          "0x0000000000000..00000000000000000000000000007"
      ]
  ],
  [
      "0xbb9bc244d798123fde783fcc1c72d3bb8c189413",
      []
  ]
]
```

- Two addresses and two storage slots are specified
- Gas cost = $2 \times 2400 + 2 \times 1900$

Type 2 Transaction Format

Type 2 Transaction Format

- Type 2 transaction
  - `0x02 || rlp([chain_id, nonce, max_priority_fee_per_gas, max_fee_per_gas, gas_limit, destination, amount, data, access_list, signature_y_parity, signature_r, signature_s])`
- Proposed in EIP 1559; included in London hard fork (Aug 2021)
- New fields
  - `max_priority_fee_per_gas`
  - `max_fee_per_gas`
- EIP 1559 introduced a base fee per block
- Every transaction has to pay the base fee which is burned
- The `max_priority_fee_per_gas` specifies a tip to the miner
- The `max_fee_per_gas` specifies the maximum value of base fee plus tip the transaction is willing to pay

References

- Yellow paper https://ethereum.github.io/yellowpaper/paper.pdf
- EIP-2718 https://eips.ethereum.org/EIPS/eip-2718
- EIP-1559 https://eips.ethereum.org/EIPS/eip-1559
- EIP-2930 https://eips.ethereum.org/EIPS/eip-2930
- EIP-1559 https://eips.ethereum.org/EIPS/eip-1559
- Understanding gas costs after Berlin https://hackmd.io/@fvictorio/gas-costs-after-berlin
- Berlin upgrade announcement https://blog.ethereum.org/2021/03/08/ethereum-berlin-upgrade-announcement
- London upgrade announcement https://blog.ethereum.org/2021/07/15/london-mainnet-announcement
- EVM Opcodes https://www.evm.codes/
- 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
- Online Keccak hash https://emn178.github.io/online-tools/keccak_256.html