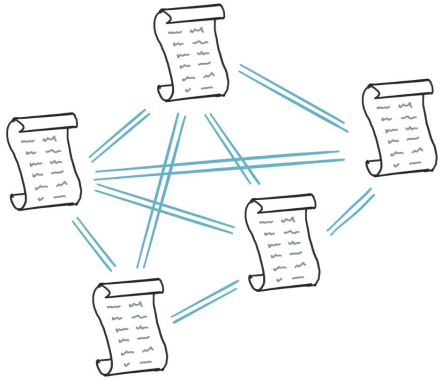
Intro to Ethereum

Ethereum Blockchain
Smart Contracts
Sample Contracts
Remix Editor

Ethereum Blockchain

The Ethereum Blockchain



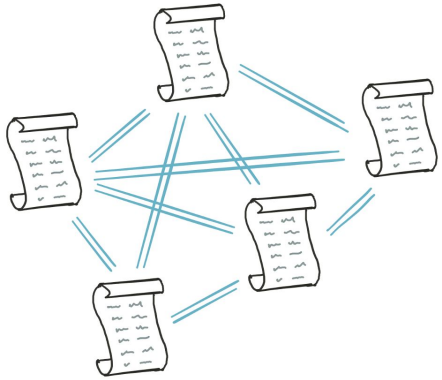
Other Blockchains / Technologies

- Bitcoin - limited script and accounts logic
- Distributed Ledger Technology - private databases
- Neo, Rootstock, etc…

Ethereum is public, immutable, decentralized application state

- Anyone can join, publish smart contracts, transact, read state
- Updates are transactional, verified by merkle proofs
- Quasi-turing complete “global computer”

Technical Details



Ether (ETH) is the settlement currency

10^{18} Wei = 1 ETH ~ $??? USD https://coinmarketcap.com/

Computation uses Gas https://ethgasstation.info/ https://converter.murkin.me/

Charging for computation: a solution to the halting problem (quasi-turing complete)

EVM: 256 bit virtual machine - stack based to a depth of 1024

Storage: every contract has it's own key:value database, unlimited if you pay for it

Languages: Solidity (JavaScript), Serpent (python), LLL (lua)

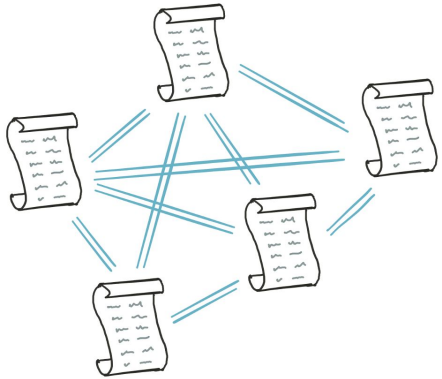
Networks: mainnet (real) - testnets - private networks

Gas Costs

Appendix G. Fee Schedule

The fee schedule G is a tuple of 31 scalar values corresponding to the relative costs, in gas, of a number of abstract operations that a transaction may effect.

Name	Value	Description*
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.
$G_{balance}$	400	Amount of gas to pay for a BALANCE operation.
G_{sload}	200	Paid for a SLOAD operation.
$G_{jumpdest}$	1	Paid for a JUMPDEST operation.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	5000	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{suicide}$	24000	Refund given (added into refund counter) for suiciding an account.
$G_{suicide}$	5000	Amount of gas to pay for a SUICIDE operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{codedeposit}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}	700	Paid for a CALL operation.
$G_{callvalue}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{callstipend}$	2300	A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer.
$G_{newaccount}$	25000	Paid for a CALL or SUICIDE operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
$G_{expbyte}$	10	Partial payment when multiplied by $\lceil \log_{256}(exponent) \rceil$ for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
G_{txcreate}	32000	Paid by all contract-creating transactions after the *Homestead transition*.
$G_{txdatazero}$	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.
$G_{transaction}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
$G_{logdata}$	8	Paid for each byte in a LOG operation's data.
$G_{logtopic}$	375	Paid for each topic of a LOG operation.
G_{sha3}	30	Paid for each SHA3 operation.
$G_{sha3word}$	6	Paid for each word (rounded up) for input data to a SHA3 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{blockhash}$	20	Payment for BLOCKHASH operation.

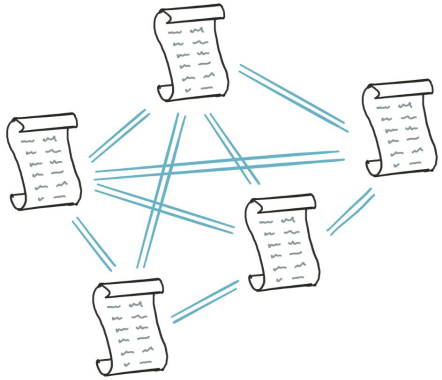


Yellow Paper

Highly Technical Overview of the EVM

http://yellowpaper.io/

Accounts and Transactions



Two different account types: EOA and Contract

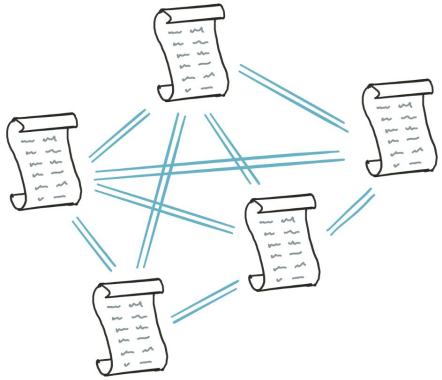
Externally Owned Accounts (EOA)

- This is your wallet (like in Bitcoin) controlled by private key
- Contains nonce, balance, storageRoot
- Transaction cost = 21,000 gas

Contract Accounts (EOA + code)

- Can contain, hold, send, restrict access to ETH
- Transactions execute code, execution costs gas
- Sending ETH to contracts could be more costly than EOA to EOA

Transactions



Transactions contain

- recipient, signature, value, gas limit, gas price, [data]

Transaction lifecycle

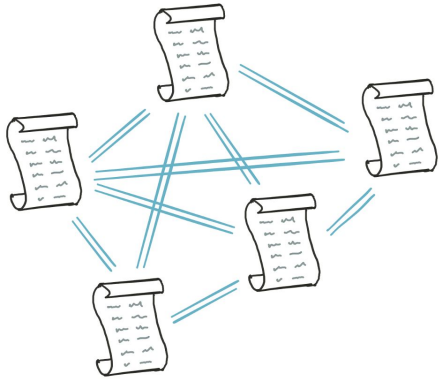
- Local / Node: created, signed, sent to node, hashed, broadcast
- All Nodes: include in block, new state / events created, mining
- Winner: broadcast, verified, update state + events, confirmation(s)

Transaction IDs are a 256bit hash of the transaction for tracking

Example

- https://etherscan.io/tx/0x93951ca835179683d5c10945e76ac585b4c5e34ed56b4e44b158f8bbea664a01

Messages



Contract to contract, virtual objects in EVM, never serialized

Messages contain

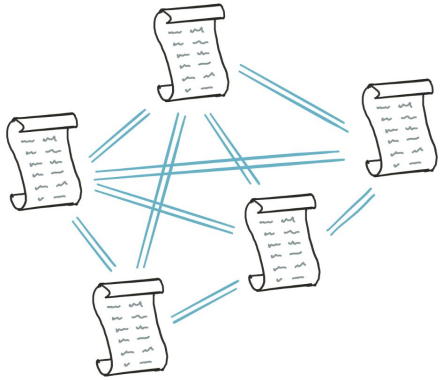
- sender, recipient, gas limit, data

Example EtherDelta Trade function

- https://etherscan.io/tx/0xbb2d0d8909138a4fe9fc7f5cde25c7107aa9c95a46d0eaf09e2545e3482bfa09

What goes in the data?

Dissecting a Function Call



```
transfer(address _to, uint256 _value)
example smart contract function
```

example function call

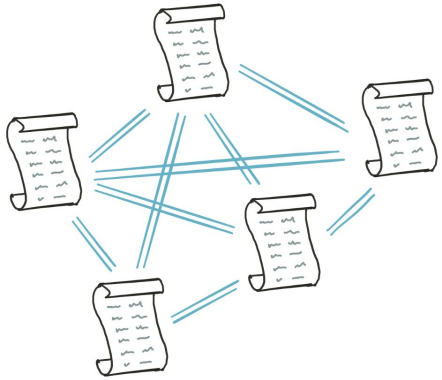
```
0xa9059cbb000000000000000000000000007adee867ea91533879d083dd47ea81f0eee3a37e0000000000000000000000000000000000000000000000000000d02ab486cedbffff
```

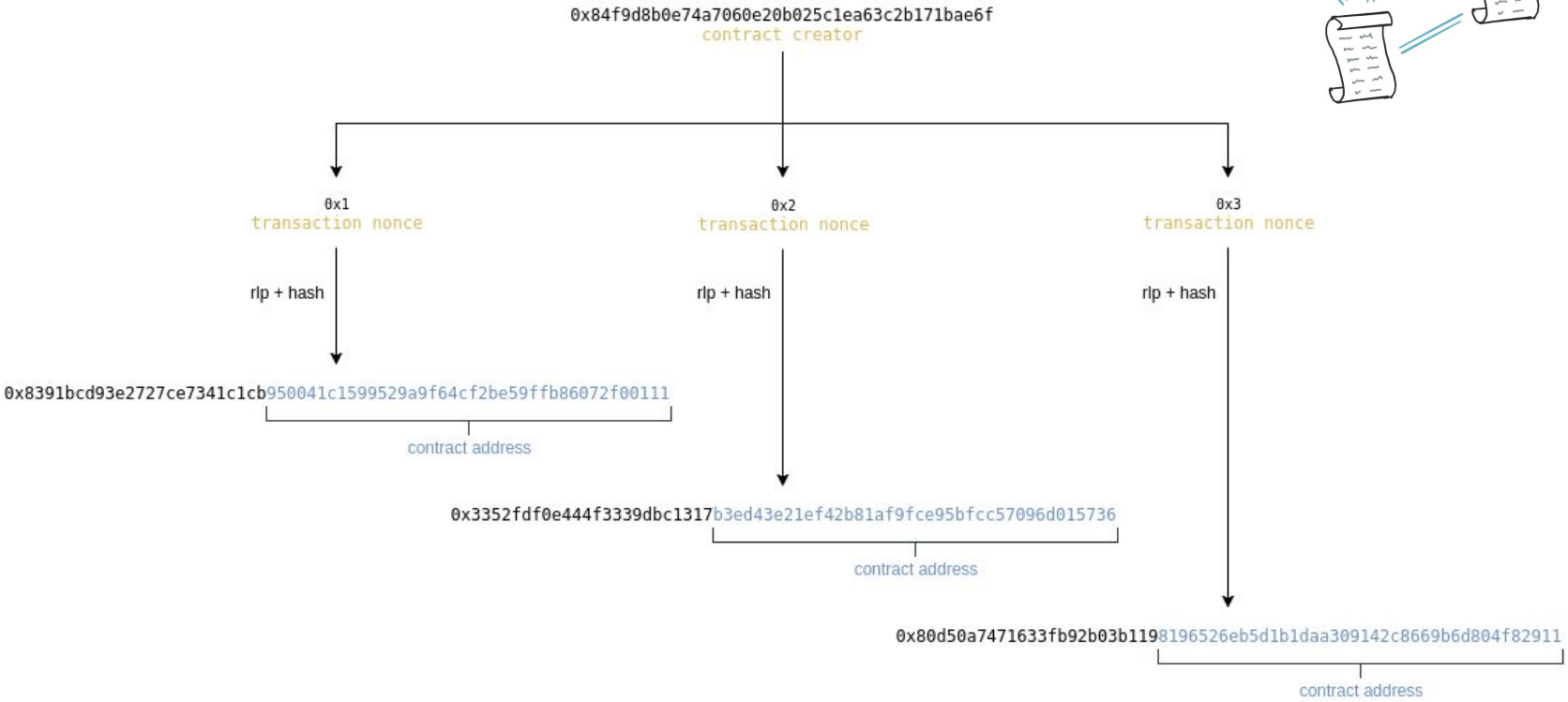
```
0x7adee867ea91533879d083dd47ea81f0eee3a37e
address _to
```

```
0x2ab486cedbffff
uint256 _value
```

```
0xa9059cbb2ab09eb219583f4a59a5d0623ade346d962bcd4e46b11da047c9049b
hash of "transfer(address,uint256)"
```

Dissecting Contract Addresses

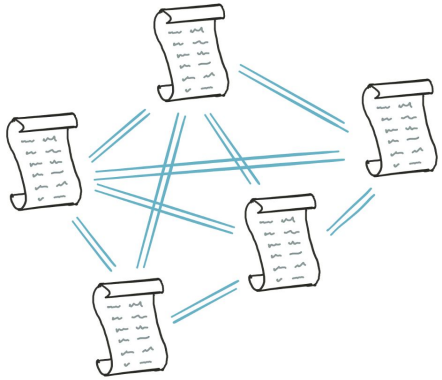




Smart Contracts

A smart contract is a computerized transaction protocol that executes the terms of a contract. The general objectives are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries. Related economic goals include lowering fraud loss, arbitrations and enforcement costs, and other transaction costs. **-Szabo '94**

Smart Contracts Overview



Code written in Solidity or other lang. and compiled

Stored on blockchain as Contract Account with EVM bytecode

Functions called by sending transactions with or without ETH

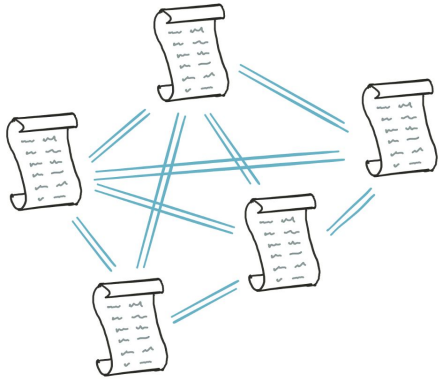
- Gas costs must be paid to execute even if no ETH is being sent to contract

Contracts can call functions and create instances of other contracts

Functions are computed and verified by miners

An ERC20 token is a single instance of a smart contract

Contract State and Networks

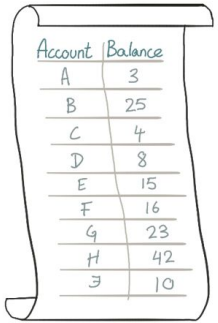


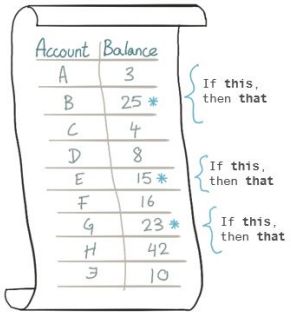
Several networks but only one “mainnet”

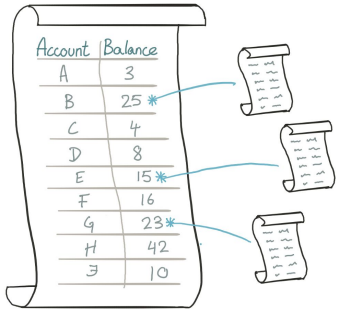
Test Networks: Ropsten, Kovan and Rinkeby

Local Testing: testrpc, remix

A deployed contract = instance with 1 network and 1 state

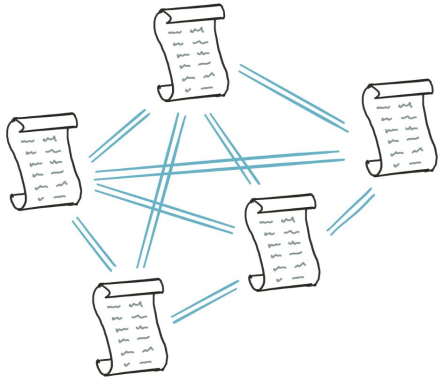






Remix Introduction

Remix Online IDE



https://remix.ethereum.org/

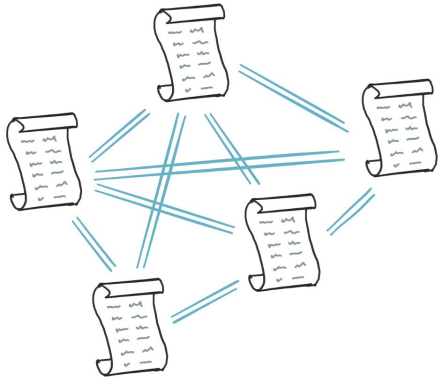
Quick way to write Solidity, compile, test and debug

Using Javascript VM

- 5 accounts, make sure to select the right one (sanity check)
- Addresses must be sent as string args

Ballot Example Walkthrough

Compiling on Remix and the ABI

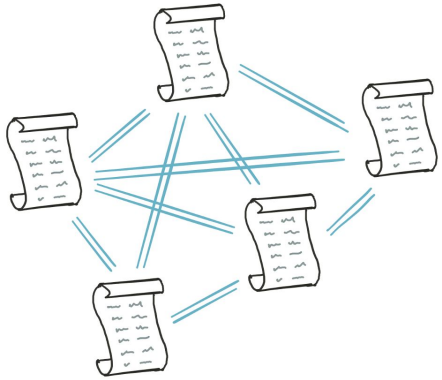


Compiling our contract produces the ABI

- ByteCode - with and without initialization
- Application Binary Interface (not blockchain)
- MetaData: compiler, source hash
- OpCodes and Assembly view

ByteCode without initialization code is the only code stored on the blockchain

Deploying from Remix

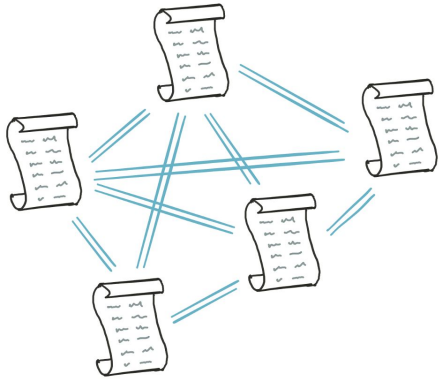


Sample:

```
var browser_hello_sol_helloContract = web3.eth.contract(ABI_GOES_HERE);
var browser_hello_sol_hello = browser_hello_sol_helloContract.new(
   {
     from: web3.eth.accounts[0],
     data: BYTECODE_GOES_HERE,
     gas: '4700000'
   }, function (e, contract){
    console.log(e, contract);
    if (typeof contract.address !== 'undefined') {
         console.log('Contract mined! address: ' + contract.address + ' transactionHash: ' +
contract.transactionHash);
    }
 })
```

MetaMask

MetaMask is a Light Client



Not **directly** connecting to Ethereum blockchain (or testnet)

- Connecting “via” an API (ferryman / infura)
- Service provided by Consensys and supporters

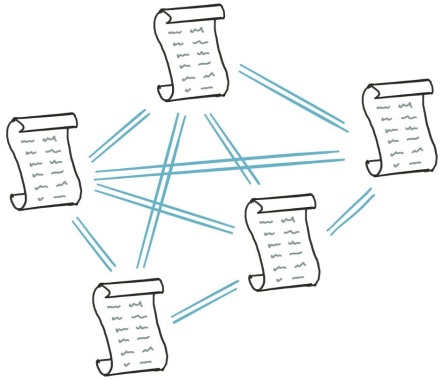
Connecting using a full node

- Run geth or another Ethereum client like parity
- Connect to --rinkeby or --kovan
- Default --testnet is ropsten

```
geth --rpc --rpcaddr --rpcport 8545
```

Let’s Play

Sample Contract Hello



```
pragma solidity ^0.4.15;

contract Hello {
  string message;
  event Talk(string _message, address _from);
  function talk(string _message) public {
     message = _message;
     Talk(_message, msg.sender);
  }
}
```

On Kovan! 0x9b735192e179d1c29ccf6305d2860f382a3bbf31

Sample Contract HelloMarket

```
pragma solidity ^0.4.15;
contract HelloMarket {
  address public owner;
  uint256 public price;
  string public message;
  function HelloMarket() public {
      owner = msg.sender;
  }
  modifier onlyOwner() {
      require(msg.sender == owner);
      _;
  }
  event Talk(string _message, address _from);
  function talk(string _message) public onlyOwner {
      message = _message;
      Talk(_message, msg.sender);
  }
  function buyRights() payable public returns (bool) {
      if (msg.value > price) {
          owner = msg.sender;
          price = msg.value;
          return true;
      }
      return false;
  }
}
```

On Kovan! [0xbc0f099204ff9f8ccec4c7c5a25d6211454ea780]

Decentralized Apps

Decentralized Applications (DApps)

User opens app that is utilizing Web3.js (library for ethereum)

Web3 loads user account and connects to smart contract(s)

App reads state

User sends transactions

Blockchain validates update

App updates state

User sees result of transaction