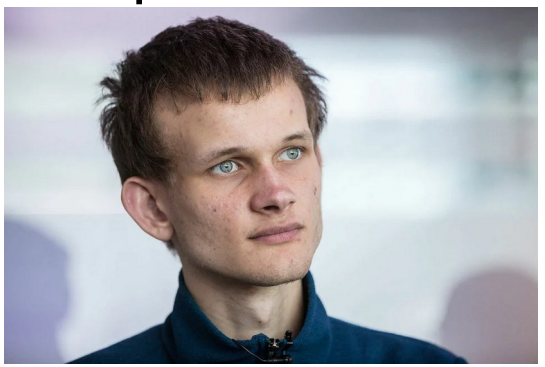
Ethereum

Opensource public blockchain with smart contracts. 15 seconds block time. Proof-of-work algorithm Ethash uses regenerating 1GB data for hashing to resist ASICs mining.
Switched to Proof-of-Stake (POS) consensus 15 sept. 2022.



Vitalik Buterin proposed architecture in 2013, raised money, hired two developer teams, founded consortium.

Main features: smart contracts using Turing-complete language, transactions run contracts and consume GAS for operations, trafic and storage, users pay for GAS using Ether token.

DAO exploit in 2016 lead to theft of $50 million worth ether caused a fork: ETH (Ethereum, theft reversed) and ETC (Ethereum Classic).

1 Ethereum = 1 000 000 000 000 000 000 WEI (10^18)

Comparison to Bitcoin

- There are no UTXOs, only “accounts” (of two varieties).

- Block time is significantly shorter; 15 second target instead of 10 minutes.

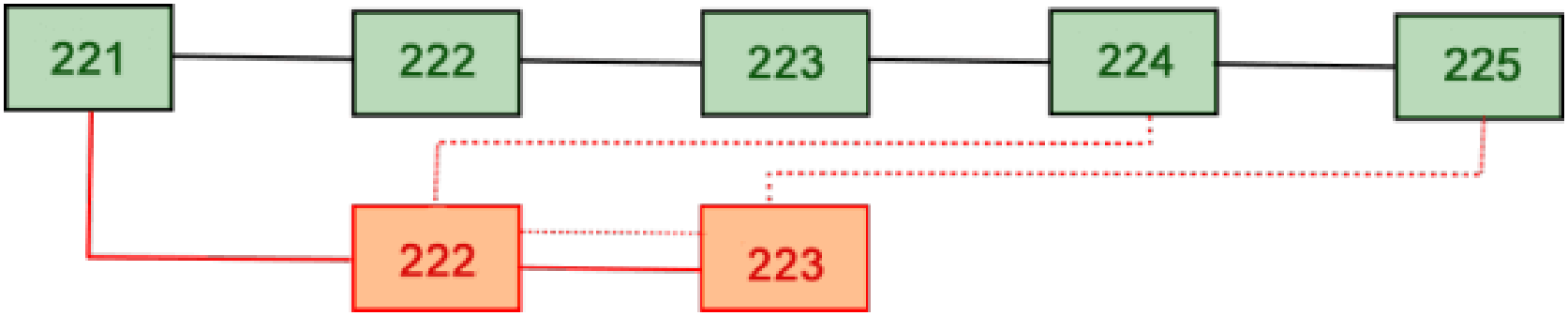
- Miners get rewards for including references to orphan blocks (so called “uncle blocks”)

- Ethereum’s scripting language is far more expressive than Bitcoin’s; typically advanced transactions/contracts are constructed using a higher level language then converted into EVM bytecode.

- Ethereum has no strict cap on monetary supply (ether supply).

- Transactions cost “gas” to run (denominated in ether) depending on how computationally expensive they are to run.

Uncle Blocks Explained



See
https://github.com/ethereum/wiki/wiki/Design-Rationale
https://github.com/ethereum/wiki/wiki/White-Paper

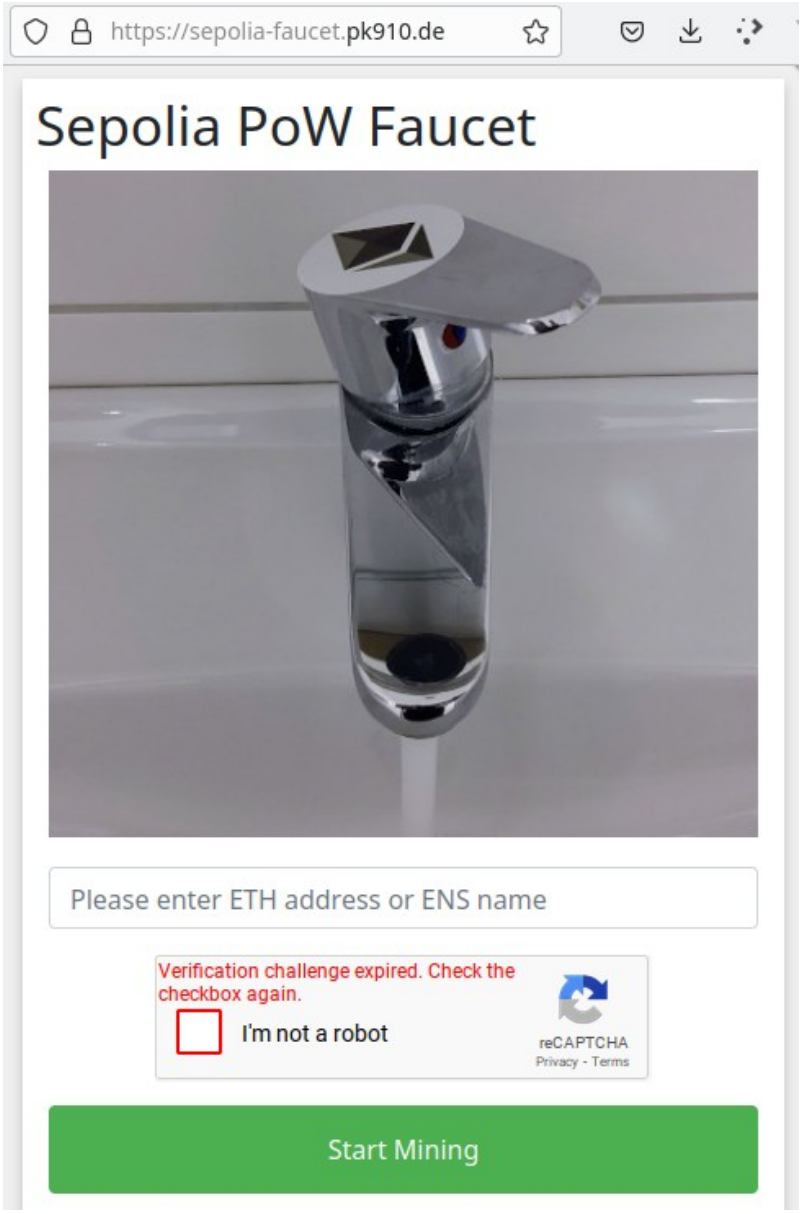
Etherium Test Networks

Sepolia:
The Sepolia network uses a permissioned validator set. It's fairly new, meaning its state and history are both quite small. This means the network is quick to sync to and that running a node on it requires less storage. This is useful for users who want to quickly spin up a node and interact with the network directly.
- Closed validator set, controlled by client & testing teams
- New testnet, less applications deployed than other testnets
- Fast to sync and running a node requires minimal disk space

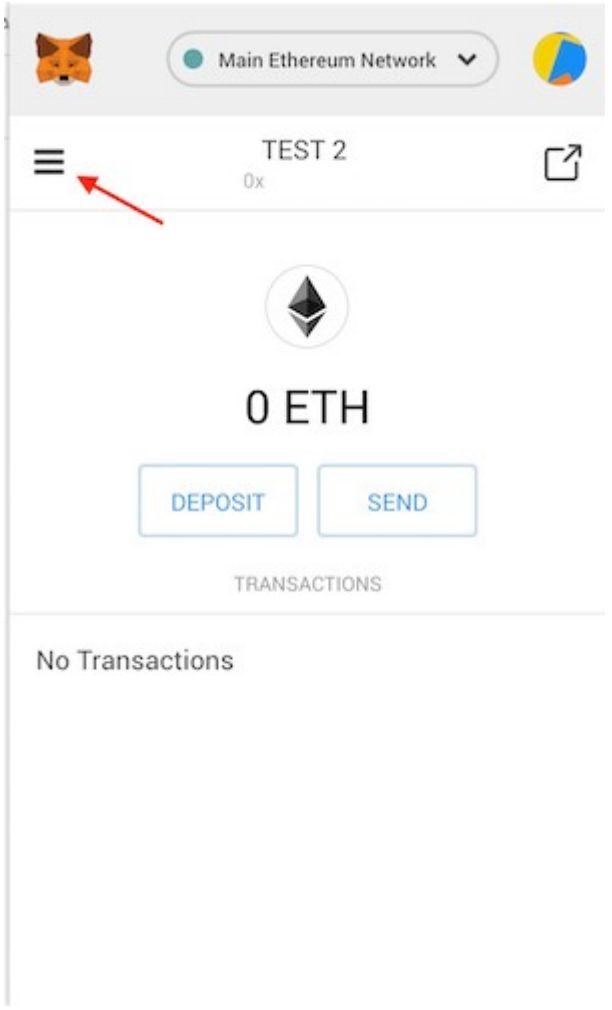
Goerli (long-term support):
Note: the Goerli testnet is deprecated(opens in a new tab)
and will be replaced by Holesovice(opens in a new tab) in 2023. Please consider migrating your applications to Sepolia.
Goerli is a testnet for testing validating and staking. The Goerli network is open for users wanting to run a testnet validator. Stakers wanting to test protocol upgrades before they are deployed to mainnet should therefore use Goerli.
- Open validator set, stakers can test network upgrades
- Large state, useful for testing complex smart contract interactions
- Longer to sync and requires more storage to run a node

https://ethereum.org/en/developers/docs/networks/

Etherium Test Networks



Metamask Google Chrome Plugin



See https://metamask.io/

Smart Contracts

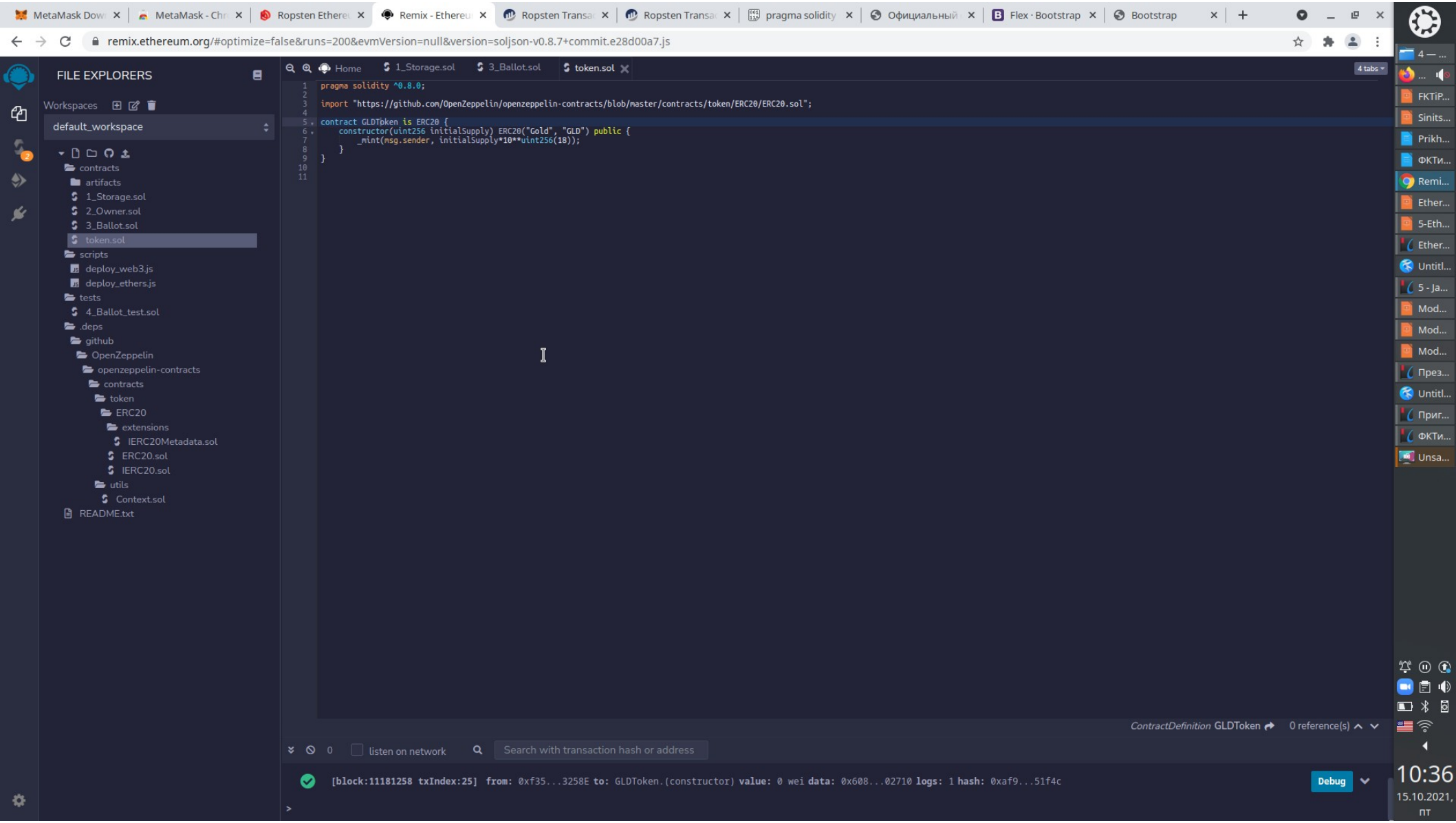
Etherium smart contracts are Decentralized Applications (DApps), used for 50%+ of Initial Coin Offerings (ICO) and most of corporate private blockchains for business needs.

Written in high level language (Solidity is the most popular), compiled to EVM byte code. Code can be public, can not be changed after it is deployed to blockchain.

Smart Contract Example Greeter.sol

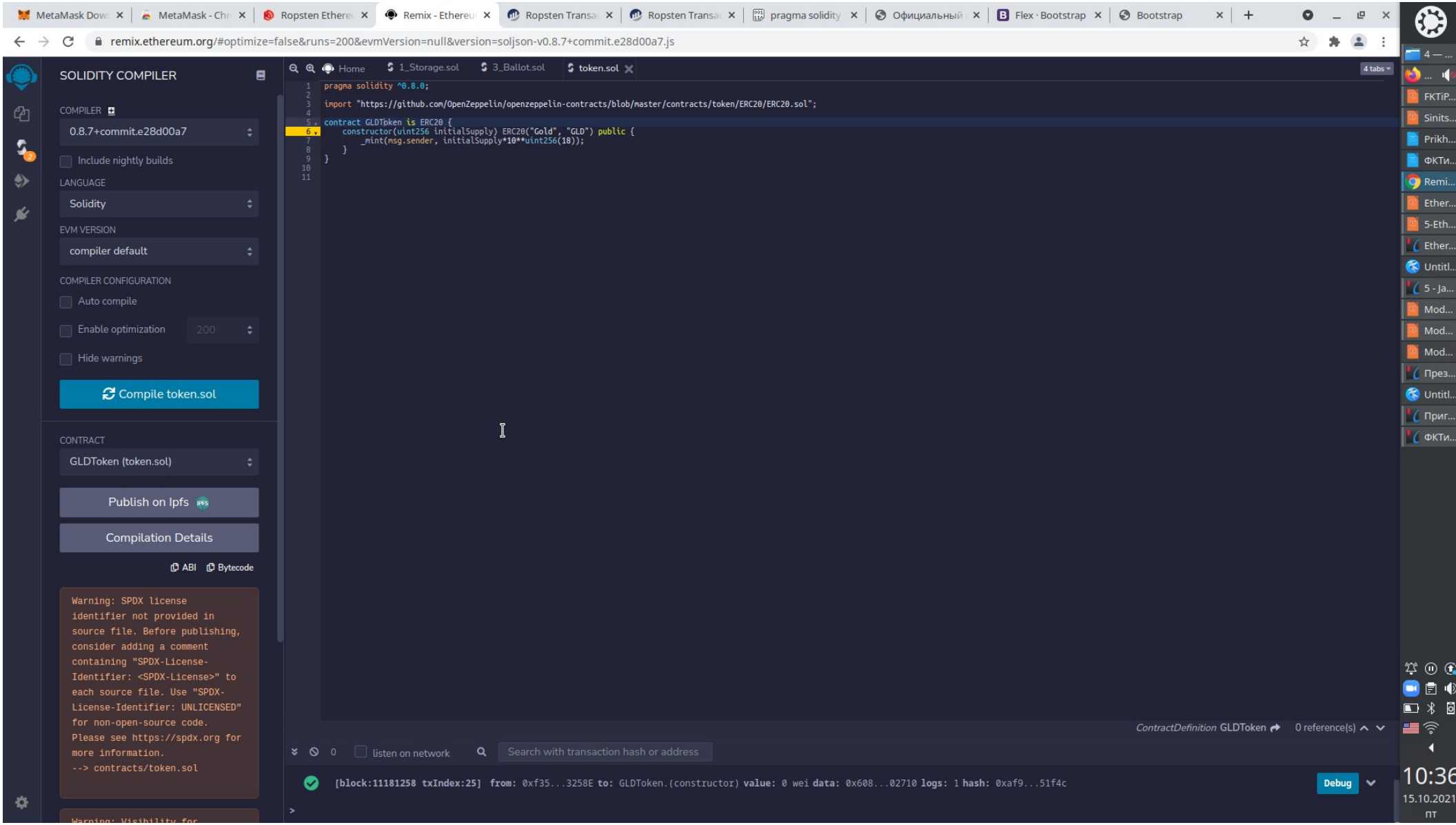
```
pragma solidity ^0.5.0;

contract Greeter {

  string greeting;

  function greet() public view returns (string memory) {
    return greeting;
  }

  function setGreet(string memory _greeting) public {
    greeting = _greeting;
  }
}
```

Remix Online IDE: develop

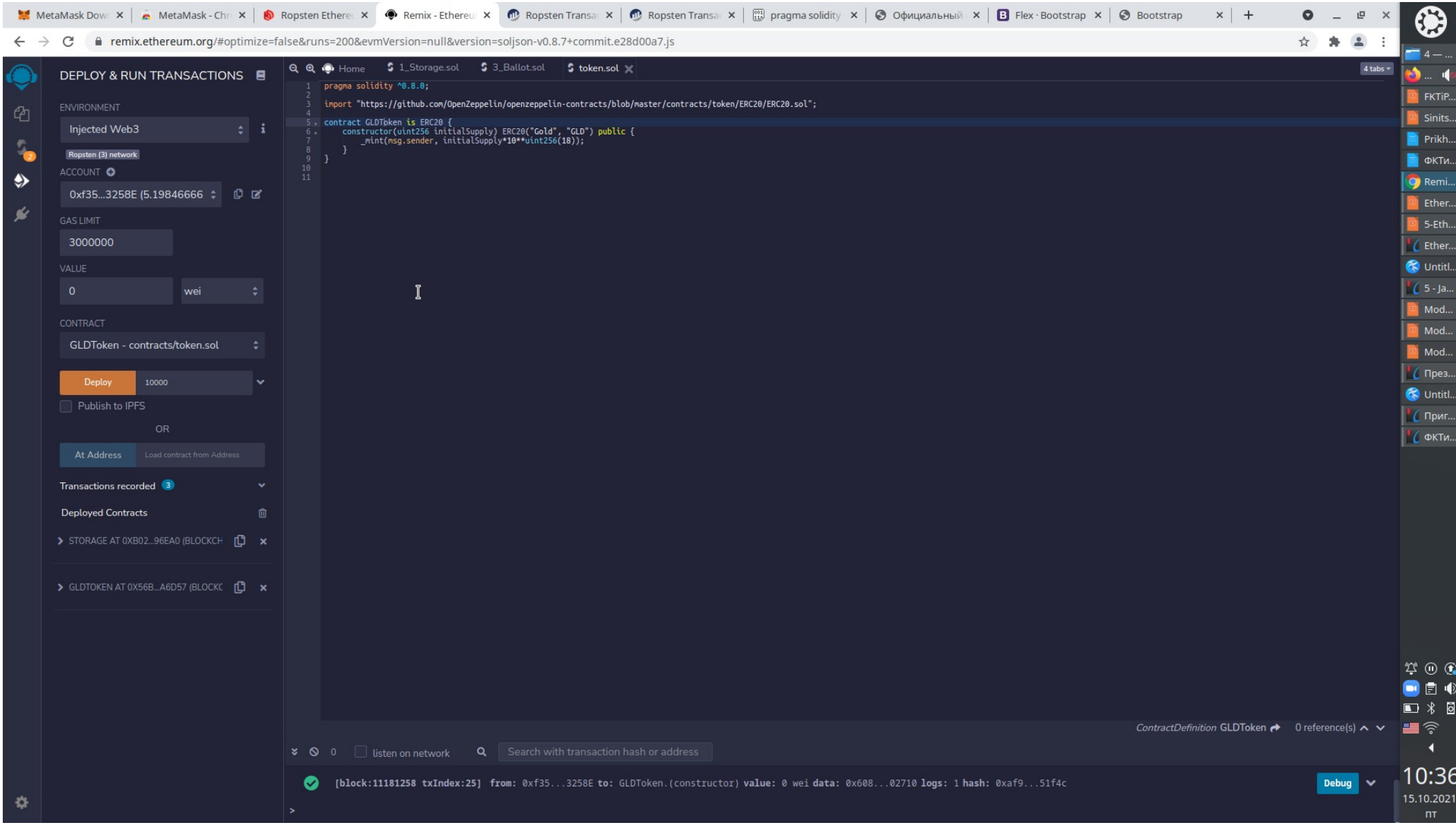


https://remix.ethereum.org/

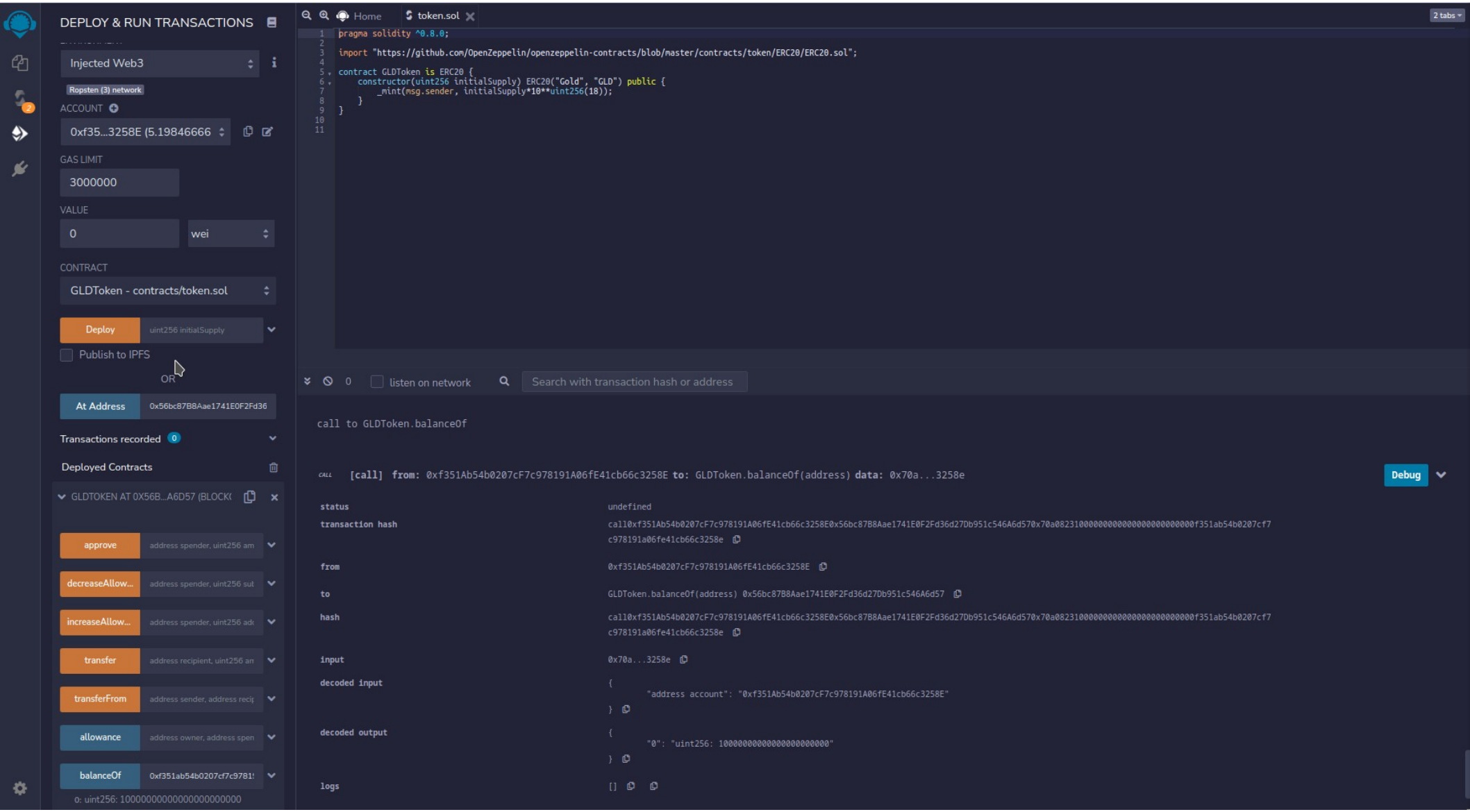
Remix Online IDE: compile



Remix Online IDE: deploy



Remix Online IDE: call



Etherium Nodes

Geth (Go) https://github.com/ethereum/go-ethereum

Parity (Rust) https://github.com/paritytech/parity-ethereum

Installing and Launching Geth in Test Networks

sudo snap install geth

Start in Ropsten network: pre-configured proof-of-work test network:
geth --syncmode light --testnet --datadir test-chain-dir --rpc --rpcaddr localhost --rpcport 3334 --rpcapi personal,eth,net,web3b,web3 console

Took 24 hours 04.2020 with --syncmode fast. Took **66GB** space!

With --syncmode light it starts immidiately without additional download or disk space.
But you should wait for 5-15 minutes to get peers connected.

By default, geth accepts connections from the loopback interface (127.0.0.1), listening port is 8545.
JSON-RPC method namespaces must be whitelisted in order to be available through the HTTP server.

See https://geth.ethereum.org/docs/interface/command-line-options

Private test blockchain

For fast development, trial and errors:

– no peers

– gas price 0

– blocks are mined automatically for faster transaction processing

– can be started multiple times with multiple data directories to simulate network

– first address added automatically gets large amount of Ether

Start private test blockchain:

geth --dev --datadir dev-chain-dir --rpc --rpcaddr localhost --rpcport 3334
--rpcapi personal,eth,net,web3b,web3 console

See https://geth.ethereum.org/getting-started/private-net

Useful Geth Console Commands

Current block number:
>eth.blockNumber
0

Syncing process start:
>eth.syncing
{
  currentBlock: 7156602,
  highestBlock: 7888574,
  knownStates: 198941293,
  pulledStates: 198941293,
  startingBlock: 7117116
}

Current peer count:
>net.peerCount
2

Default address:
>eth.coinbase
"0xd6e6d7270d84c361bcd91047da65cbfc01157230"

See https://geth.ethereum.org/docs/interface/command-line-options

Geth Sync Modes

Full Sync: It gets the block headers, the block bodies, and validates every element from genesis block. A full Geth node processes the entire blockchain and replays all transactions that ever happened. Disk and CPU intensive!

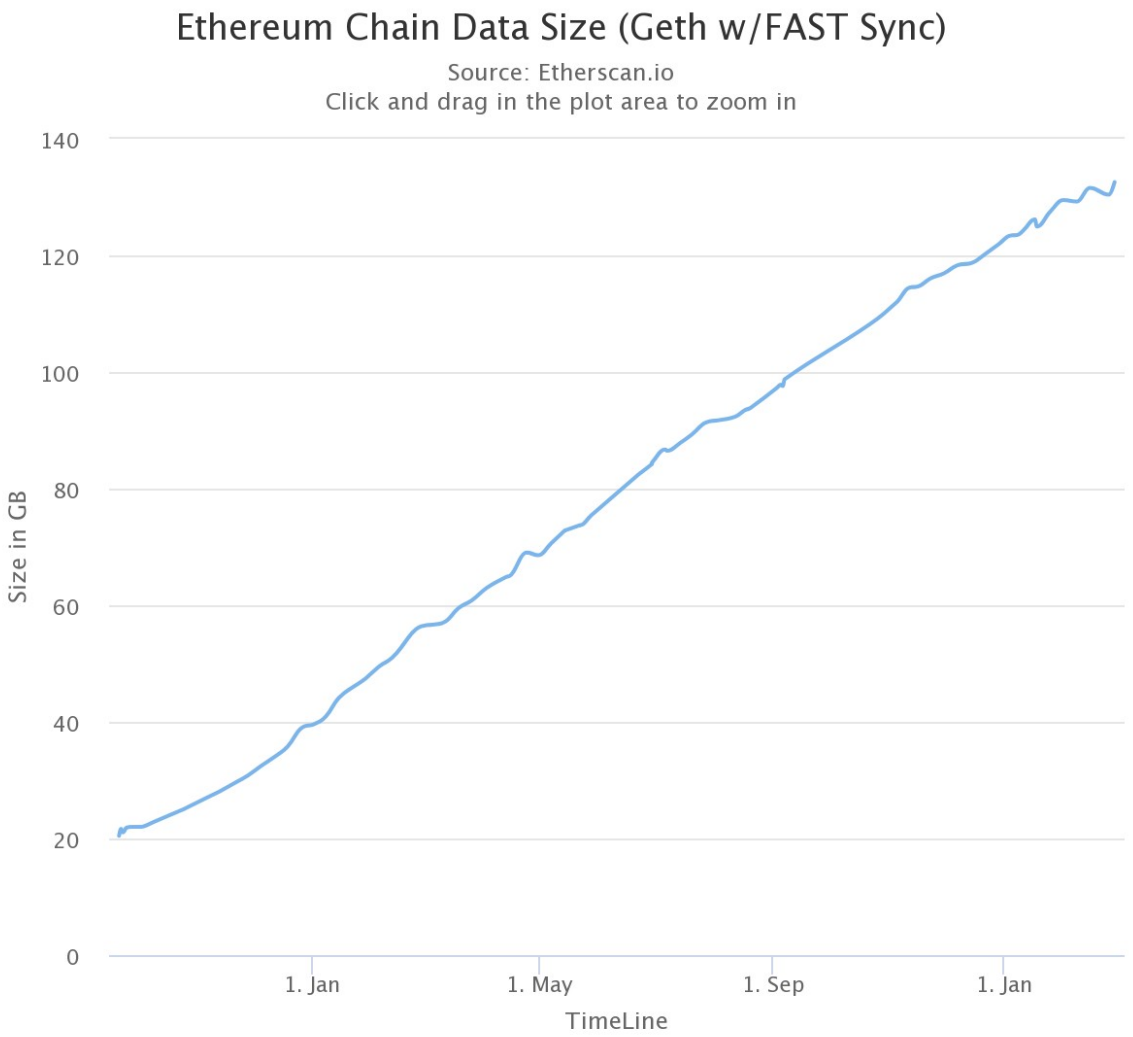
i3.2xlarge AWS EC2 instances ($0.624 per Hour, 8 cores, 61 GiB RAM, ephemeral volume 1.9 TiB NVMe SSD) 11 hours, 176GiB space

Fast Sync: It gets the block headers, the block bodies, it processes no transactions until current block. Then it gets a snapshot state and goes like a full synchronization. A fast Geth node downloads all transaction receipts in parallel to all blocks, and the most recent state database. It switches to a full synchronization mode once done with that. Note, that this results not only in a fast sync but also in a pruned state-database because the historical states are not available for blocks smaller than best block minus 1024.

Light Sync: It gets only the current state. Light node download just the header of the blocks and that too it does not download all the blocks. For all kind of processing, light node depends on full node peers.

See
https://etherworld.co/2018/03/13/understanding-ethereum-light-node/
https://dev.to/5chdn/the-ethereum-blockchain-size-will-not-exceed-1tb-anytime-soon-58a
https://www.chaindata.club/

Ethereum Chain Data Size



Source https://etherscan.io/chart2/chaindatasizefast

Geth Add Account (Debian)

> personal.newAccount("123") ← super secret password!

INFO [04-03|13:53:52.119] Your new key was generated
address=0x97a4f1fd5FF01b5CF056cC5C1FB44b2ABBDB99Cb

WARN [04-03|13:53:52.119] Please backup your key file!
path=/home/sin/Univer/Blockchain/Ethereum/test-chain-dir/keystore/UTC--2020-04-03T10-53-50.158448651Z--97a4f1fd5ff01b5cf056cc5c1fb44b2abbdb99cb

WARN [04-03|13:53:52.119] Please remember your password!
"0x97a4f1fd5ff01b5cf056cc5c1fb44b2abbdb99cb"

eth.accounts
["0xd6e6d7270d84c361bcd91047da65cbfc01157230",
"0x97a4f1fd5ff01b5cf056cc5c1fb44b2abbdb99cb"]

> exit

See https://geth.ethereum.org/docs/interface/managing-your-accounts
https://github.com/ethereum/go-ethereum/wiki/Managing-your-accounts

Geth JSON RPC client

Request:

curl --url http://localhost:3334/ -H "Content-type:application/json" -X POST --data '{"jsonrpc":"2.0","method":"web3_clientVersion","params":[],"id":67}'

Response:

{"jsonrpc":"2.0","id":67,"result":"Geth/v1.9.6-unstable/linux-amd64/go1.11.13"}

See full specs:

https://github.com/ethereum/wiki/wiki/JSON-RPC

NodeJS Geth Web3 client

```
npm install web3
./node

>var Web3 = require('web3');
>var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:3334"));
>web3.eth.personal.getAccounts().then(console.log);

[ '0xd6e6D7270d84C361BCd91047da65CBfC01157230',
  '0x97a4f1fd5FF01b5CF056cC5C1FB44b2ABBDB99Cb' ]

>web3.eth.personal.newAccount().then(console.log);
Promise {
  <pending>,
…

>'0x123456789101112131415161718192021222345 6'

>web3.eth.getBalance('0xd6e6D7270d84C361BCd91047da65CBfC01157230').then(console.log);

> 1157920892373161954235709850086879078532699846656405640394575840079131296399 27

>
web3.utils.fromWei("1157920892373161954235709850086879078532699846656405640394575840079131296399 27", "ether");
'1157920892373161954235709850086879078532699846656405640394 57.5840079131296399 27'
```

Full specs https://web3js.readthedocs.io/en/v1.2.6/

NodeJS Parity Web3 client

```
>var Web3 = require('web3');
>var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));

>web3.eth.personal.newAccount("123").then(console.log);
Promise {
  <pending>,
…

>0x33C8578BCFac70C0d6b469f284C38635cd1970a5

>web3.eth.getBalance('0xd6e6D7270d84C361BCd91047da65CBfC01157230').then(console.log);

> 1157920892373161954235709850086879078532699846656405640394575840079131296399 27

>
web3.utils.fromWei("1157920892373161954235709850086879078532699846656405640394575840079131296399 27", "ether");
'1157920892373161954235709850086879078532699846656405640394 57.5840079131296399 27'
```

Full specs https://web3js.readthedocs.io/en/v1.2.6/

NodeJS Geth Web3 transaction

```
web3.eth.sendTransaction({from: '0xd6e6d7270d84c361bcd91047da65cbfc01157230',
to: '0x97a4f1fd5ff01b5cf056cc5c1fb44b2abbdb99cb',
value: 10000000000000000000}).then(console.log);
```

```
Promise {
  <pending>,
  domain:
   Domain {
     domain: null,
     _events:
      [Object: null prototype] {
        removeListener: [Function: updateExceptionCapture],
        newListener: [Function: updateExceptionCapture],
        error: [Function: debugDomainError] },
     _eventsCount: 3,
     _maxListeners: undefined,
     members: [] } }
> { blockHash:
   '0xcf254bfcdda692960c96d16a92c6711479c4931e6ec012d9eac0999e96fac5c2',
  blockNumber: 1,
  contractAddress: null,
  cumulativeGasUsed: 21000,
  from: '0xd6e6d7270d84c361bcd91047da65cbfc01157230',
  gasUsed: 21000,
  logs: [],
  logsBloom:
   '0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000',
  status: true,
  to: '0x97a4f1fd5ff01b5cf056cc5c1fb44b2abbdb99cb',
  transactionHash:
   '0x82fa526b5037b9e9336fd31c3bbdbf15d220e00d144114f0a58dcd7cb57f2be2',
  transactionIndex: 0 }
```

NodeJS Geth Web3 transaction

web3.eth.sendTransaction({from: '0x97a4f1fd5ff01b5cf056cc5c1fb44b2abbdb99cb',
to: '0xa48897460bc178b50d1de04983eb0ec904664d6c',
value: 50000000000000000}).then(console.log);

Error: Returned error: authentication needed: password or unlock

We may not unlock addres via HTTP, only from geth console.

Geth keystore

Geth stores keys in keystore folder. By default it is located in data directory.

Each key in separate file in encrypted form:

Filename example:
UTC—2020-04-03T10-53-50.158448651Z--97a4f1fd5ff01b5cf056cc5c1fb44b2abbdb99cb

File content:
{"address":"97a4f1fd5ff01b5cf056cc5c1fb44b2abbdb99cb","crypto":{"cipher":"aes-128-ctr","ciphertext":"4d05abf6d7d4643b6fdf4673d7e2d0c57b9f9c30e05de02b3c759ca214b3ae7b","cipherparams":{"iv":"774b79612177d40d2ca8b9fe628983aa"},"kdf":"scrypt","kdfparams":{"dklen":32,"n":262144,"p":1,"r":8,"salt":"1b637672bd011400c839f813ad2a1dd6719b7c6aa67b840c38a31adec54ebf61"},"mac":"7b5b798fa661fac6c62e1a9b8b2d19b5df56353b44ea24a468aa2854396668c8"},"id":"2795f164-955c-474d-9fec-4316564b856e","version":3}

A library to work with Ethereum private keys

See https://github.com/ethereumjs/keythereum

```
var keythereum = require("keythereum");
var datadir = "./dev-chain-dir";
var address= "0x97a4f1fd5ff01b5cf056cc5c1fb44b2abbdb99cb";
const password = "123";

var keyObject = keythereum.importFromFile(address, datadir);
var privateKey = keythereum.recover(password, keyObject);
console.log(privateKey.toString('hex'));
```

NodeJS code to send signed transaction via Web3 Example

```
const Web3 = require('web3');


// Connect to local Geth node or any other peer
const web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:3334"));


// Contents of keystore file, can do a fs read
const keystore = '{"address":"97a4f1fd5ff01b5cf056cc5c1fb44b2abbdb99cb","crypto":{"cipher":"aes-128-
ctr","ciphertext":"4d05abf6d7d4643b6fdf4673d7e2d0c57b9f9c30e05de02b3c759ca214b3ae7b","cipherparams":
{"iv":"774b79612177d40d2ca8b9fe628983aa"},"kdf":"scrypt","kdfparams":
{"dklen":32,"n":262144,"p":1,"r":8,"salt":"1b637672bd011400c839f813ad2a1dd6719b7c6aa67b840c38a31adec54ebf61"},"mac":"7b5b
798fa661fac6c62e1a9b8b2d19b5df56353b44ea24a468aa2854396668c8"},"id":"2795f164-955c-474d-9fec-
4316564b856e","version":3}';


const decryptedAccount = web3.eth.accounts.decrypt(keystore, '123'); // Provide password used for key encryption


const rawTransaction = {
  "from": "0x97a4f1fd5ff01b5cf056cc5c1fb44b2abbdb99cb", // Keystore account id
  "to": "0xa48897460bc178b50d1de04983eb0ec904664d6c", // Account you want to transfer to
  "value": web3.utils.toHex(web3.utils.toWei("0.5", "ether")),
  "gas": 21000, // Current Ethereum gas fee per transaction
  "chainId": 1337 // Geth --dev network id
};


decryptedAccount.signTransaction(rawTransaction)
  .then(signedTx => web3.eth.sendSignedTransaction(signedTx.rawTransaction))
  .then(receipt => console.log("Transaction receipt: ", receipt))
  .catch(err => console.error(err));


// Or sign using private key from decrypted keystore file or keythereum recovered private key
// web3.eth.accounts.signTransaction(rawTransaction, decryptedAccount.privateKey).then(console.log);
```

NodeJS code to send signed transaction via Web3 Output

```
Transaction receipt:  { blockHash:
   '0xa206e18ed852890a852e95f49cd309278e0a27eac86d29fdc1d5b1484c93d9fb',
  blockNumber: 2,
  contractAddress: null,
  cumulativeGasUsed: 21000,
  from: '0x97a4f1fd5ff01b5cf056cc5c1fb44b2abbdb99cb',
  gasUsed: 21000,
  logs: [],
  logsBloom:

'0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000',
  status: true,
  to: '0xa48897460bc178b50d1de04983eb0ec904664d6c',
  transactionHash:
   '0xe6b2709ec0e1882ff8ba373c47025800305917 96b39619a53656ab89929883e4',
  TransactionIndex: 0 }
```

Geth console results:

```
>eth.getBalance("0x97a4f1fd5ff01b5cf056cc5c1fb44b2abbdb99cb");

499999999999979000

>eth.getBalance("0xa48897460bc178b50d1de04983eb0ec904664d6c");

500000000000000000

>eth.getTransactionCount('0x97a4f1fd5ff01b5cf056cc5c1fb44b2abbdb99cb')

1
```

Smart Contracts Solidity compiler SOLC

Installation:
snap install solc

Usage:
solc Greeter.sol --combined-json abi,bin --pretty-json > solc-output.json

Documentation:
https://solidity.readthedocs.io/en/develop/using-the-compiler.html

Solc-output.json


```
{
  "contracts":
  {
    "Greeter.sol:Greeter":
    {
      "abi": "[{\"constant\":true,\"inputs\":[],\"name\":\"greet\",\"outputs\":
[{\"internalType\":\"string\",\"name\":\"\",\"type\":\"string\"}],\"payable\":false,\"stateMutability\":\"view\",\"type\":\"function\"},
{\"constant\":false,\"inputs\":[{\"internalType\":\"string\",\"name\":\"_greeting\",\"type\":\"string\"}],\"name\":\"setGreet\",\"outputs\":
[],\"payable\":false,\"stateMutability\":\"nonpayable\",\"type\":\"function\"}]",
      "bin":
"608060405234801561001057600080fd5b5061030f806100206000396000f3fe608060405234801561001057600080fd5b5060043610
610036576000356e01c80639698086b1461003b578063cfae3217146100f6575b600080fd5b6100f460048036036020811015610051
57600080fd5b810190808035906020019064010000000081111561006e57600080fd5b820183602082011115610080576000fd5b80
359060200191846001830284011164010000000083111715610a257600080fd5b91908080601f016020809104026020016040519081
016040528093929190818152602001838380828437600081840152601f19601f820116905080830192505050505050509192919290
505050610179565b005b6100fe610193565b604051808060200182810382528381815181526020019150805190602001908083836000
005b83811015610161e578082015181840152602081019050610123565b505050509050909081019060 1f16801561016b578082038051
6001836020036101000a031916815260200191505b509250505060405180910390f35b806000908051906020019061018f929190610
235565b5050565b606060008054600181600116156101000203166002900480601f016020809104026020016040519081016040528
092919081815260200182805460018160011615610100020316600290048015610 22b5780601f106102005761010080835404028352
9160200191610 22b565b820191906000526020600020905b81548152906001019060200180831161020e57829003601f168201915
b505050505090509056 5b828054600181600116156101000203166002900490600052602060002090601f016020900481019282601f
106102765780516 0ff1916838001178555610 2a4565b828001600101855582156102a4579182015b828111156102a35782518255916
020019190600101906102885 65b5b5090506102b191906102b5565b5090565b6102d791905b808211156102d357600081600090555
06001016102bb565b5090565b9056fea265627a7a723158204faeb8d17e118c130ed115c8a0ef9ca5350bd48cce17a51c792516e529c
594f164736f6c63430005100032"
    }
  },
  "version": "0.5.16+commit.9c3226ce.Linux.g++"
}
```

Deploy Contract

```
const fs = require("fs");
const Web3 = require('web3');

// Create a web3 connection to a running geth node over JSON-RPC running at
// http://localhost:3334
const web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:3334"));

// Read the compiled contract code
// Compile with
// solc Greeter.sol --combined-json abi,bin --pretty-json > solc-output.json
let source = fs.readFileSync("solc-output.json");
let contracts = JSON.parse(source)["contracts"];

// ABI description as JSON structure
let abi = JSON.parse(contracts["Greeter.sol:Greeter"].abi);

// Smart contract EVM bytecode as hex
let code = '0x' + contracts["Greeter.sol:Greeter"].bin;

var myContract = new web3.eth.Contract(abi);

myContract.deploy({
    data: code,
    //arguments: [123, 'My String']
})
.send({
    from: '0xd6e6d7270d84c361bcd91047da65cbfc01157230', // The address the transaction should be sent from.
    gas: 1500000, // (optional): The maximum gas provided for this transaction (gas limit).
    gasPrice: '30000000000000'  // (optional): The gas price in wei to use for this transaction.
}, function(error, transactionHash){ })
.on('error', function(error){ })
.on('transactionHash', function(transactionHash){ })
.on('receipt', function(receipt){
   console.log(receipt.contractAddress) // contains the new contract address
})
.on('confirmation', function(confirmationNumber, receipt){  })
.then(function(newContractInstance){
    console.log(newContractInstance.options.address) // instance with the new contract address
});
```

Deploy Contract Output

Output:
0x047048435756729cf734c4beED0e1f5D4D2a410C

Geth console autput:
> INFO [05-13|01:42:32.460] Submitted contract creation
fullhash=0x43f21f639ac010d4d35f2fbe8637c75f1c4c22570e3f358348a3179b1687d618
contract=0x047048435756729cf734c4beED0e1f5D4D2a410C
INFO [05-13|01:42:32.460] Commit new mining work number=3 sealhash=93f329…decc67
uncles=0 txs=1 gas=222254 fees=6.66762 elapsed=577.21µs
INFO [05-13|01:42:32.462] Successfully sealed new block number=3 sealhash=93f329…decc67
hash=8f5323…ae8099 elapsed=1.187ms
INFO [05-13|01:42:32.462] mined potential block number=3 hash=8f5323…ae8099
INFO [05-13|01:42:32.462] Commit new mining work number=4 sealhash=15017a…1dd168
uncles=0 txs=0 gas=0 fees=0 elapsed=492.629µs

See https://web3js.readthedocs.io/en/v1.2.0/web3-eth-contract.html#contract-deploy

Contract Call Method Example

```
const fs = require("fs");
const Web3 = require('web3');

// Create a web3 connection to a running geth node over JSON-RPC running at
// http://localhost:3334
const web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:3334"));

// Read the compiled contract code
// Compile with
// solc Greeter.sol --combined-json abi,bin --pretty-json > solc-output.json
let source = fs.readFileSync("solc-output.json");
let contracts = JSON.parse(source)["contracts"];

// ABI description as JSON structure
let abi = JSON.parse(contracts["Greeter.sol:Greeter"].abi);

var myContract = new web3.eth.Contract(abi, '0x047048435756729cf734c4beED0e1f5D4D2a410C');

// Will call a “constant” method and execute its smart contract method in the EVM without sending any
transaction.
// Note calling can not alter the smart contract state.
// Will output empty string.
myContract.methods.greet().call()
  .then(console.log);
```

Contract Call Method Output

Output:
<empty string>

Geth console autput:
<none>

Contract Send Transaction Example

```
...

// Will send a transaction to the smart contract and execute its method.
// Note this can alter the smart contract state.
myContract.methods.setGreet("Hello World!").send({
  from: '0xd6e6d7270d84c361bcd91047da65cbfc01157230', // The address the transaction should be sent from, eth.coinbase dev address used .
  gas: 1500000, // (optional): The maximum gas provided for this transaction (gas limit).
  gasPrice: '30000000000000'  // (optional): The gas price in wei to use for this transaction.
})
.on('transactionHash', function(hash){
  console.log("hash");
  console.log(hash);
})
.on('receipt', function(receipt){
  console.log("receipt");
  console.log(receipt);
})
.on('confirmation', function(confirmationNumber, receipt){
  console.log("confirmation");
  console.log(receipt);
})
.on('error', console.error);
```

Contract Send Transaction Output

Output:

hash
0x398014b18bb6126adc15589b161995a391b6730e6b74c363a24a04bd907b5322

receipt
{ blockHash:
'0xc5723c4023796a4e3c911b3913d68fea4ae51a0c19778d93b2490cc615f3234c',
blockNumber: 18,
contractAddress: null,
cumulativeGasUsed: 29157,
from: '0xd6e6d7270d84c361bcd91047da65cbfc01157230',
gasUsed: 29157,
logsBloom:
'0x0...',
status: true,
to: '0x047048435756729cf734c4beed0e1f5d4d2a410c',
transactionHash:
'0x398014b18bb6126adc15589b161995a391b6730e6b74c363a24a04bd907b5322',
transactionIndex: 0,
events: {} }

confirmation
{ blockHash:
'0xc5723c4023796a4e3c911b3913d68fea4ae51a0c19778d93b2490cc615f3234c',
... }

Geth console output:

> INFO [05-13|12:46:55.957] Submitted transaction
fullhash=0x398014b18bb6126adc15589b161995a391b6730e6b74c363a24a04bd907b5322
recipient=0x047048435756729cf734c4beED0e1f5D4D2a410C
INFO [05-13|12:46:55.957] Commit new mining work number=18 sealhash=797955…02e99f uncles=0 txs=1 gas=29157
fees=0.87471 elapsed=361.625µs
INFO [05-13|12:46:55.958] Successfully sealed new block number=18 sealhash=797955…02e99f hash=c5723c…f3234c
elapsed=685.707µs
INFO [05-13|12:46:55.958] mined potential block number=18 hash=c5723c…f3234c
INFO [05-13|12:46:55.958] Commit new mining work number=19 sealhash=a7a837…373b86 uncles=0 txs=0 gas=0 fees=0
elapsed=415.867µs
INFO [05-13|12:46:55.959] Sealing paused, waiting for transactions

Contract Send Transaction Side Effect

Contract Call Method now prints:
Hello World!

New value is stored in a World State of your smart contract in Ethereum Blockchain!

Contract Send Transaction from Address Example

```
// Combine send signed transaction and send cotract transaction examples.

var myContract = new web3.eth.Contract(abi, '0x047048435756729cf734c4beED0e1f5D4D2a410C');
const mydata = myContract.methods.setGreet("Hi!").encodeABI();

const rawTransaction = {
  //"nonce": 21, // Result of eth.getTransactionCount('0x047048435756729cf734c4beED0e1f5D4D2a410C')
+ 1
  "from": "0x97a4f1fd5ff01b5cf056cc5c1fb44b2abbdb99cb", // Decrypted account address
  "to": "0x047048435756729cf734c4beED0e1f5D4D2a410C", // Account of smart contract instance
  "value": "0x00", // Method is not payable
  //"gasLimit": web3.utils.toHex(5000000),
  //"gasPrice": web3.utils.toHex(web3.utils.toWei('10', 'gwei')),
  //"gasPrice": '300000000000',
  "gas": 400000, // Gas limit
  "chainId": 1337, // Geth --dev network id
  "data": mydata
};

decryptedAccount.signTransaction(rawTransaction)
  .then(signedTx => web3.eth.sendSignedTransaction(signedTx.rawTransaction))
  .then(receipt => console.log("Transaction receipt: ", receipt))
  .catch(err => console.error(err));
```

Contract Send Transaction from Address Output And side effect

Side effect is that smart contract now stores:
Hi!

Output is almost the same as previous Send Transaction but now gas price is deduced from our Decrypted Account!

Pending and Queued transactions

Pending transactions are waiting to be mined in next block. Should be mined automatically in geth —dev mode.

Queued transactions are transactions where the transaction nonce is not in sequence. They will be pending automatically after the transaction with lesser noonces are mined.

The transaction nonce is an incrementing number for each transaction with the same From address. Skip noonce in —dev mode for geth to calculate transaction nonce correctly.

Tools To Debug

Geth console commands:
eth.blockNumber – returns last block number
eth.getBlock(number) – returns block details
eth.getBlock("latest")
eth.pendingTransactions – list of pending transactions
eth.getTransactionReceipt(txhash) – transaction details
eth.getTransactionCount(address) – number of transactions for addres or contract
eth.getTransactionCount(address, "pending") – get correct transaction count
eth.gasPrice – current gas price

txpool.status – number of pending and queued transactions in memory pool
txpool.inspect – list all network transactions waiting to be mined
txpool.content – detailed list all network transactions waiting to be mined

Transaction input data decoders, for example:
https://lab.miguelmota.com/ethereum-input-data-decoder/example/

Transaction encoders:
https://abi.hashex.org/

Ethereum remote node via Web3 API:
https://infura.io

Additional Libraries

https://github.com/ethereumjs/ethereumjs-tx – advanced transaction builder

https://github.com/ethereumjs/ethereumjs-abi – ABI encoder/decoder

https://github.com/ethereumjs/keythereum – working with Geth keys from JS

Links

https://geth.ethereum.org/docs/

https://web3js.readthedocs.io/

Book:

'Ethereum for Web Developers' by Santiago Palladino (https://www.apress.com/gp/book/9781484252772)

Homework

1. Install Metamask.

2. Install Geth.

3. Launch Geth it in -dev mode.

4. Optionally launch it in test network to syncronize in fast or light mode (requires up to 70 GB space).