UNIVERSITY *of* NICOSIA

Session 2

Bitcoin Core

BLOC 514: Emerging Topics in Blockchain and Digital Currency

Session Objectives

Bitcoin Core

- The fundamental notations for Bitcoin Core
- How to get network information
- How to use Command Line Interface (CLI)
- How to explore transactions
- Understand the basic features of latest release

Agenda

Bitcoin Core

Bitcoin Core: Introduction

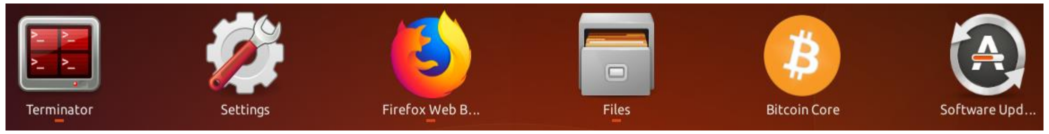
- **Bitcoin Core**: implementation of Bitcoin core
  - Initially, developed by Satoshi Nakamoto
  - Currently, supported by a team of developers: https://github.com/bitcoin/bitcoin
- Main functionality at a glance
  - Execute transactions verification
  - Connect to the Bitcoin network
  - Manage wallet that enables the transfer of bitcoins
- Main programs
  - **Bitcoin Core** (formerly Bitcoin-Qt): Graphical User Interface (GUI) front-end
  - **bitcoind**: daemon, i.e., command line program that runs in the background
  - **bitcoin-cli**: command line program interface for interacting with bitcoind (client)
    - Via remote procedure call (RPC)-based commands
    - The responses of bitcoind are in JSON format

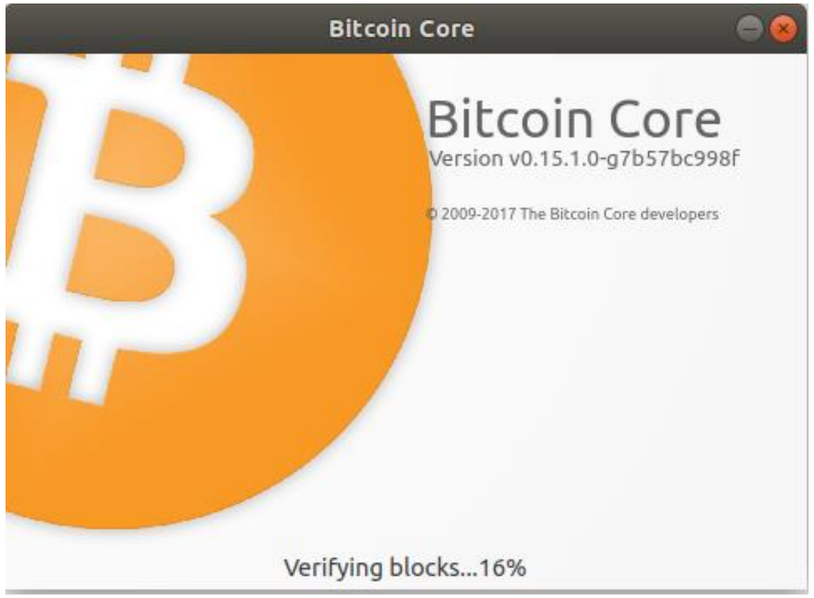
Bitcoin Installation (not required for the course; optional)

- You can download the appropriate Bitcoin installer from https://bitcoin.org/en/download or you can find more options here
  - For Windows, Mac OSX and Ubuntu Linux the installation is straightforward.
  - For different Linux flavors compilation may be required.
    If your Linux-based machine runs an appropriate graphical desktop environment and the necessary software development pre-requisites are installed, then both bitcoind and Bitcoin-Qt can be compiled on it.
- After installation you will have to wait until the initial synchronization of the entire Bitcoin blockchain is done, which may take, depending on your bandwidth and the number of connected Bitcoin nodes, several days.
  - Testnet exhibits less time/space requirements compared to mainnet
- Work in testnet mode, so that no real currency is at stake!

Bitcoin Core

- In Linux Ubuntu
- After successful installation, see the Bictcoin Core icon in “Show Applications”.
- Once started, the following screen appears (note the “Verifying blocks … %” message)
- *Note: similar functionality for other OS*



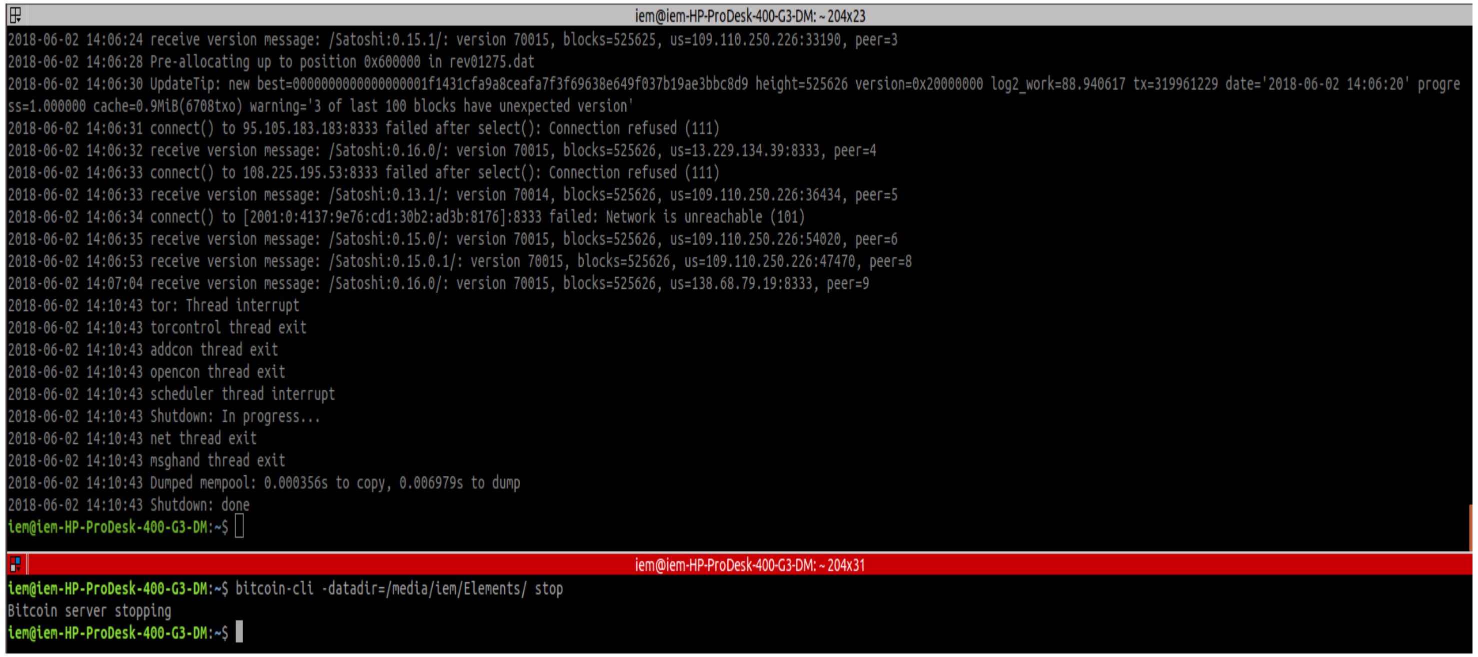

Bitcoin Core

Bitcoin Core
Version v0.15.1.0-g7b57bc998f
© 2009-2017 The Bitcoin Core developers

Verifying blocks...16%

Bitcoin Core: Behind the front-end

- Command line interface in Linux Ubuntu
- bitcoind: deamon running in the background
- bitcoin-cli: client



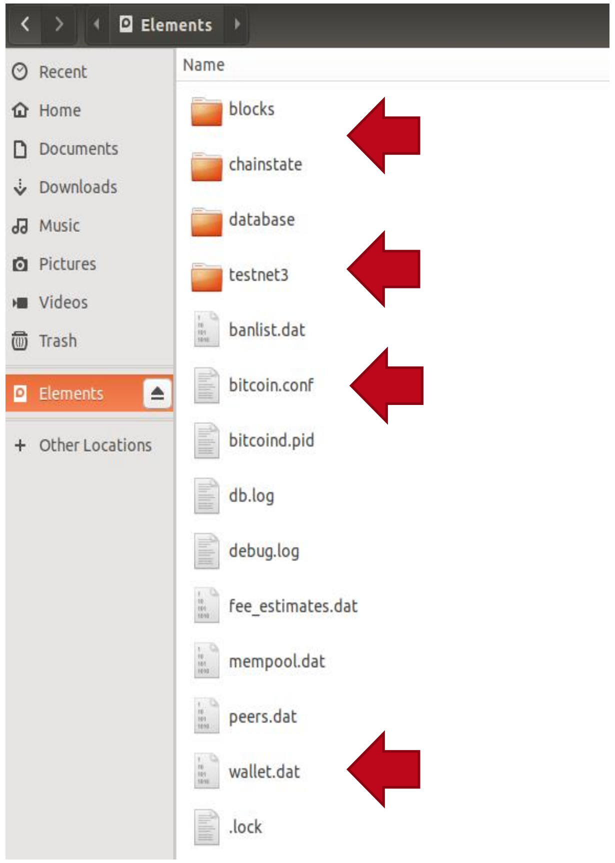
```
2018-06-02 14:06:24 receive version message: /Satoshi:0.15.1/: version 70015, blocks=525625, us=109.110.250.226:33190, peer=3
2018-06-02 14:06:28 Pre-allocating up to position 0x600000 in rev01275.dat
2018-06-02 14:06:30 UpdateTip: new best=000000000000000001f1431cfa9a8ceafa7f3f69638e649f037b19ae3bbc8d9 height=525626 version=0x20000000 log2_work=88.940617 tx=319961229 date='2018-06-02 14:06:20' progre
ss=1.000000 cache=0.9MiB(6708txo) warning='3 of last 100 blocks have unexpected version'
2018-06-02 14:06:31 connect() to 95.105.183.183:8333 failed after select(): Connection refused (111)
2018-06-02 14:06:32 receive version message: /Satoshi:0.16.0/: version 70015, blocks=525626, us=13.229.134.39:8333, peer=4
2018-06-02 14:06:33 connect() to 108.225.195.53:8333 failed after select(): Connection refused (111)
2018-06-02 14:06:33 receive version message: /Satoshi:0.13.1/: version 70014, blocks=525626, us=109.110.250.226:36434, peer=5
2018-06-02 14:06:34 connect() to [2001:0:4137:9e76:cd1:30b2:ad3b:8176]:8333 failed: Network is unreachable (101)
2018-06-02 14:06:35 receive version message: /Satoshi:0.15.0/: version 70015, blocks=525626, us=109.110.250.226:54020, peer=6
2018-06-02 14:06:53 receive version message: /Satoshi:0.15.0.1/: version 70015, blocks=525626, us=109.110.250.226:47470, peer=8
2018-06-02 14:07:04 receive version message: /Satoshi:0.16.0/: version 70015, blocks=525626, us=138.68.79.19:8333, peer=9
2018-06-02 14:10:43 tor: Thread interrupt
2018-06-02 14:10:43 torcontrol thread exit
2018-06-02 14:10:43 addcon thread exit
2018-06-02 14:10:43 opencon thread exit
2018-06-02 14:10:43 scheduler thread interrupt
2018-06-02 14:10:43 Shutdown: In progress...
2018-06-02 14:10:43 net thread exit
2018-06-02 14:10:43 msghand thread exit
2018-06-02 14:10:43 Dumped mempool: 0.000356s to copy, 0.006979s to dump
2018-06-02 14:10:43 Shutdown: done
iem@iem-HP-ProDesk-400-G3-DM:~$
```

```
iem@iem-HP-ProDesk-400-G3-DM:~$ bitcoin-cli -datadir=/media/iem/Elements/ stop
Bitcoin server stopping
iem@iem-HP-ProDesk-400-G3-DM:~$
```

Bitcoin Core: local files

- All files stored in the folder selected during installation (in this example, in a dedicated external HDD in Linux Ubuntu)

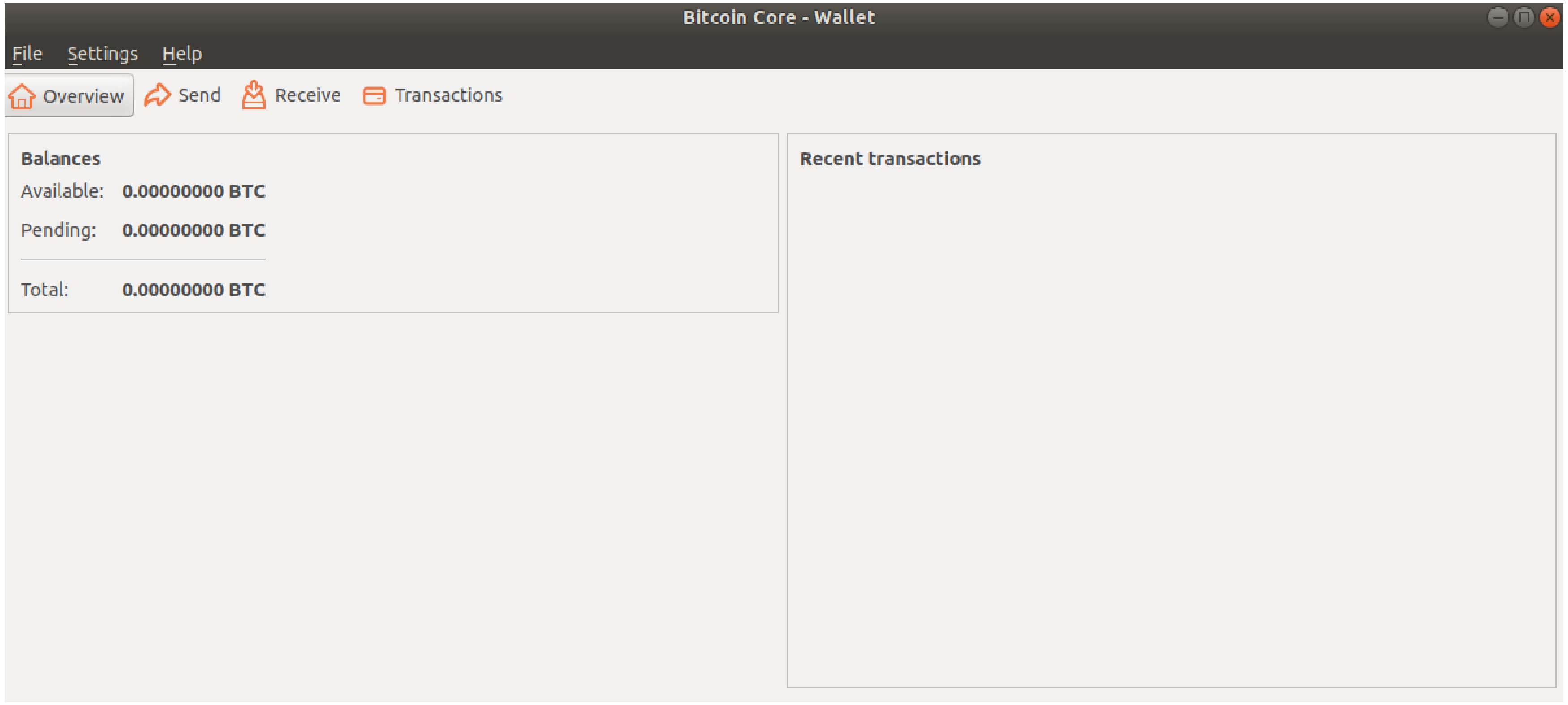
- bitcoin.conf file: configuration file
- wallet.dat file: Bitcoin wallet of mainnet
- blocks directory: blocks of mainnet
- chainstate directory: chain of blocks of mainnet
- testnet3 directory: all testnet files and directories



blocks
chainstate
database
testnet3
banlist.dat
bitcoin.conf
bitcoind.pid
db.log
debug.log
fee_estimates.dat
mempool.dat
peers.dat
wallet.dat
.lock

Bitcoin – Command Line Interface (CLI)

- Bitcoind offers a number of commands to use from the command line. The same commands can be used through the Bitcoin Core GUI (screenshot from Linux Ubuntu installation)



Balances

Available:	0.00000000 BTC
Pending:	0.00000000 BTC
Total:	0.00000000 BTC

Recent transactions

Command Line Interface (CLI)

- The complete reference to the Bitcoin client Application Programming Interface (API) can be found here: https://en.bitcoin.it/wiki/Original_Bitcoin_client/API_calls_list
- Using the command line interface you can:
  - Get information about the status of the Bitcoin network
  - Manage your wallet
  - Explore and decode transactions
  - Explore blocks
  - Create, sign and submit transactions with unspent outputs

Bitcoin Core – Getting network information

Commands to use:

Command	Description
getconnectioncount	Returns the number of connections to other nodes.
getpeerinfo	Returns data about each connected node.
getdifficulty	Returns the proof-of-work difficulty as a multiple of the minimum difficulty.
getblockcount	Returns the number of blocks in the longest block chain.
getmininginfo	Returns an object containing mining-related information:
	• blocks • currentblocksize • currentblocktx • difficulty • errors • generate • genproclimit • hashespersec • pooledtx • testnet

Managing your wallet (CLI)

Command	Parameters	Description
getnewaddress	[account]	Returns a new bitcoin address for receiving payments. If [account] is specified payments received with the address will be credited to [account].
dumpprivkey	<bitcoinaddress>	Reveals the private key corresponding to <bitcoinaddress>
importprivkey	<bitcoinprivkey> [label] [rescan=true]	Adds a private key (as returned by dumpprivkey) to your wallet. This may take a while, as a rescan is done, looking for existing transactions. Note: There's no need to import public key, as in ECDSA (unlike RSA) this can be computed from the private key.
getreceivedbyaddress	<bitcoinaddress> [minconf=1]	Returns the amount received by <bitcoinaddress> in transactions with at least [minconf] confirmations. It correctly handles the case where someone has sent to the address in multiple transactions. Keep in mind that addresses are only ever used for receiving transactions. Works only for addresses in the local wallet, external addresses will always show 0.

Managing your wallet (CLI)

Command	Parameters	Description
listtransactions	[account] [count=10] [from=0]	Returns up to [count] most recent transactions skipping the first [from] transactions for account [account]. If [account] not provided it'll return recent transactions from all accounts.
encryptwallet	<passphrase>	Encrypts the wallet with <passphrase>
walletlock		Removes the wallet encryption key from memory, locking the wallet. After calling this method, you will need to call walletpassphrase again before being able to call any methods which require the wallet to be unlocked.
walletpassphrase	<passphrase> <timeout>	Stores the wallet decryption key in memory for <timeout> seconds.
walletpassphrasechange	<oldpassphrase> <newpassphrase>	Changes the wallet passphrase from <oldpassphrase> to <newpassphrase>.

Managing your wallet (CLI)

Command	Parameters	Description
gettransaction	<txid>	Returns an object about the given transaction containing: • "amount" : total amount of the transaction • "confirmations" : number of confirmations of the transaction • "txid" : the transaction ID • "time" : time associated with the transaction. • "details" - An array of objects containing: • "account" • "address" • "category" • "amount" • "fee"
getrawtransaction	<txid> [verbose=0]	Returns raw transaction representation for given transaction id.
decoderawtransaction	<hex string>	Produces a human-readable JSON object for a raw transaction

Exploring Transactions (CLI)

```
gettransaction 0408953d670d0f268b70f449772bd3d1c1b80c690be6f3017e3f9bdaa01c0b6c
{
"amount" : -0.01080000,
"fee" : 0.01080000,
"confirmations" : 0,
"txid" : "0408953d670d0f268b70f449772bd3d1c1b80c690be6f3017e3f9bdaa01c0b6c",
"walletconflicts" : [
],
"time" : 1399871859,
"timereceived" : 1399871859,
"details" : [
{
"account" : "",
"address" : "1Dima5vfScYn342c7SfcX2pFYSu3rqhtKz",
"category" : "send",
"amount" : -0.00500000,
"fee" : 0.01080000
},
{
"account" : "",
"address" : "18Z6bDrD4Fce8hRW5DkQ68f23xBCFodyxv",
"category" : "send",
"amount" : -0.01080000,
"fee" : 0.01080000
},
{
"account" : "Dima",
"address" : "1Dima5vfScYn342c7SfcX2pFYSu3rqhtKz",
"category" : "receive",
"amount" : 0.00500000
}],……
}
```

Transaction IDs (txid) are not authoritative until a transaction has been confirmed. Absence of a transaction hash in the blockchain does not mean the transaction was not processed.

This is known as "***transaction malleability***", as transaction hashes can be modified prior to confirmation in a block. After confirmation, the txid is immutable and authoritative.

Exploring Transactions (CLI)

```
getrawtransaction
0408953d670d0f268b70f449772bd3d1c1b80c690be6f3017e3f9bdaa01c0b
6c

0100000000029181c1d7b6b4fc7e2f1f1ee43dfeb778468292a7b49a3e26c9c7
0b268fbc9ade000000006c493046022100cc2a0d920c154014d4e7f9387830
7b1b5aeab8bba25288e1f00573fe05cf8aac022100b8269506e5c431d55bbe
4c6850e983db8b9d9016cdece8dd7555a4ebf22816ba012102c8515f4e0512
378032d44d5ed3888bcd50be103ee26e0279f52a1fb935bb8f71ffffffff0e
4d456390086dd622ce8be50672de7943d2a1d0ee78593a6b4e5c7a9cb6c9c3
000000008a47304402200f9e6e9bacd1f0d44525265455e92014faba5931a0
ee8517664777d38c090d95022000905089d5bfcf8509589984f9b79182ea1b
cbf6e6ae16f765efd8390f3c835201410468190 1c41fe94cabc8e809ca1f83
0fd6bc953d88254337db8ab1db9448ecd8bb2fec05f74f38abb05f4fd5d704
0f9c011365967c24672514c2a40f20dde07094ffffffff0220a10700000000
001976a9148b87c4f4c177a46de7d50b7dd9840c16caa4728088acc07a1000
00000000001976a91452dadb8a8948da05040672a11eacaecd916aa39288ac00
000000
```

- *The gettransaction* command returns a transaction in a simplified form.
- To retrieve the full transaction code we can use two commands:
  - *getrawtransaction* and
  - *decoderawtransaction.*
- *The getrawtransaction* command uses the transaction ID as a parameter and returns the full transaction as a “raw” hex string, exactly as it is on the Bitcoin network.

Exploring Transactions (CLI)

```
decoderawtransaction 0100000002918 1c1d7b6b4fc7e2f1f1ee43dfeb…
```

```
{
"txid" : "0408953d670d0f268b70f449772bd3d1c1b80c690be6f3017e3f9bdaa01c0b6c",
"version" : 1,
"locktime" : 0,
"vin" : [
{
"txid" : "de9abc8f260bc7c9263e9ab4a792824678b7fe3de41e1f2f7efcb4b6d7c18191",
"vout" : 0,
"scriptSig" : {
"asm" :
"3046022100cc2a0d920c154014d4e7f93878307b1b5aeab8bba25288e1f00573fe05cf8aac02
2100b8269506e5c431d55bbe4c6850e983db8b9d9016cdece8dd7555a4ebf22816ba01
02c8515f4e0512378032d44d5ed3888bcd50be103ee26e0279f52a1fb935bb8f71",
"hex" :
"493046022100cc2a0d920c154014d4e7f93878307b1b5aeab8bba25288e1f00573fe05cf8aac
022100b8269506e5c431d55bbe4c6850e983db8b9d9016cdece8dd7555a4ebf22816ba012102c
8515f4e0512378032d44d5ed3888bcd50be103ee26e0279f52a1fb935bb8f71"
},
"sequence" : 4294967295
},
{
"txid" : "c3c9b69c7a5c4e6b3a5978eed0a1d24379de7206e58bce22d66d089063454d0e",
"vout" : 0,
"scriptSig" : {
"asm" : "304402200f9e6e9bacd1f0d44525265455e92014faba5931a0ee8517664777d38c0
```

- The *decoderawtransaction* command shows all the parts of this transaction, including the transaction inputs and outputs.

Exploring Transactions (CLI)

```
"vin" : [
{
"txid" :
"de9abc8f260bc7c9263e9ab4a792824678b7fe3de41e1f2f7efcb4b6d7c18191",
"vout" : 0,
"scriptSig" : {
"asm" :
"3046022100cc2a0d920c154014d4e7f93878307b1b5aeab8bba25288e1f00573fe05c
f8aac022100b8269506e5c431d55bbe4c6850e983db8b9d9016cdece8dd7555a4ebf22
816ba01
02c8515f4e0512378032d44d5ed3888bcd50be103ee26e0279f52a1fb935bb8f71",
"hex" : "49304…"
},
"sequence" : 4294967295
},
{
"txid" :
"c3c9b69c7a5c4e6b3a5978eed0a1d24379de7206e58bce22d66d089063454d0e",
"vout" : 0,
"scriptSig" : {
"asm" :
"304402200f9e6e9bacd1f0d44525265455e92014faba5931a0ee8517664777d38c090
d95022000905089d5bfcf8509589984f9b79182ea1bcbf6e6ae16f765efd8390f3c835
201
04681901c41fe94cabc8e809ca1f830fd6bc953d88254337db8ab1db9448ecd8bb2fec
05f74f38abb05f4fd5d7040f9c011365967c24672514c2a40f20dde07094",
"hex" : "4730…"
},
"sequence" : 4294967295
}
],
```

- This transaction has two inputs as outputs of previously confirmed transactions with IDs starting with *de9a* and *c3c9* respectively

Exploring Transactions (CLI)

```
"vout" : [
{
"value" : 0.00500000,
"n" : 0,
"scriptPubKey" : {
"asm" : "OP_DUP OP_HASH160
8b87c4f4c177a46de7d50b7dd9840c16caa47280 OP_EQUALVERIFY
OP_CHECKSIG",
"hex" : "76a9148b87c4f4c177a46de7d50b7dd9840c16caa4728088ac",
"reqSigs" : 1,
"type" : "pubkeyhash",
"addresses" : [
"1Dima5vfScYn342c7SfcX2pFYSu3rqhtKz"
]
```

- and one output of 5 mBits to the *1Dima...* address.

Exploring Transactions (CLI)

```
gettransaction
0408953d670d0f268b70f449772bd3d1c1b80c690be6f3017e3f9bdaa01c0b6c
{
"amount" : -0.01080000,
"fee" : 0.01080000,
"confirmations" : 2,
"blockhash" :
"000000000000000002320499cc4e60f5a515a03b088925f78b728bdf79ed5ac86",
"blockindex" : 189,
"blocktime" : 1399872120,
"txid" : "0408953d670d0f268b70f449772bd3d1c1b80c690be6f3017e3f9bdaa01c0b6c",
"walletconflicts" : [
],
"time" : 1399871859,
"timereceived" : 1399871859,
"details" : [
{
…
```

- Once the transaction is confirmed, by inclusion in the block, the *gettransaction* command returns additional information, showing the block hash (identifier) and block index for the block in which the transaction was included.

Multi-signature transactions

- Bitcoin has a feature called *"multi-signature"*, in which a transaction must have multiple independent approvals before the funds can be spent. **Multi-signature (or *"multisig"*)** transactions prevent thieves from stealing the contents of a wallet by simply gaining access to a single key-pair.

- The most common scheme for multi-signature transactions is to employ an *"M-of-N scheme"*, for instance, **2-of-3**, in which case, at least 2 people (i.e. signers) must approve a transaction. The way this works is that 3 *public keys* are listed as potential signers of a transaction, and at least 2 of those must be used to create a transaction's signature in order to spend it.

- There is currently a limitation of 15 public keys that can be used to sign a transaction, and any combination of the *"M-of-N scheme"* may be used.

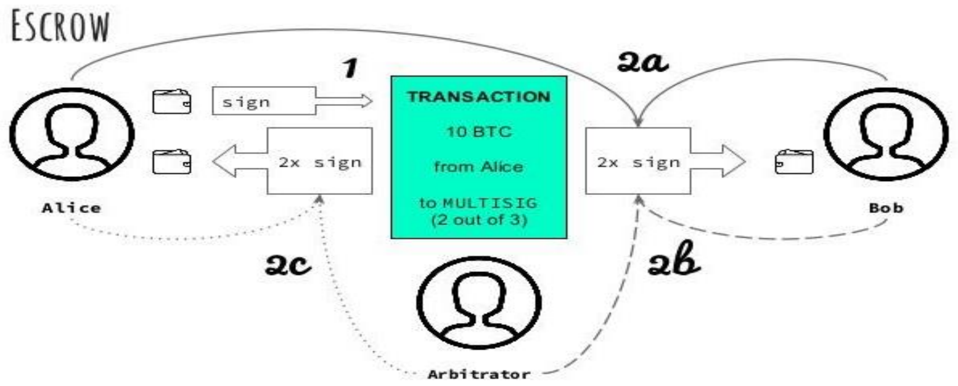
Multi-signature transactions: A problem …

The Problem

- Alice wants to buy online from Bob
- Alice doesn't want to pay until after Bob ship
- Bob doesn't want to ship until after Alice pays

Multi-signature transactions: A problem …and it's solution

- A solution in Bitcoin is to introduce a third party and do an escrow transaction.
- Escrow transactions can be implemented using MULTISIG.
- In 2-of-3 multisig :
  1. Find a relatively trustworthy escrowr (Arbitrator).
  2. Create a multisig address requiring 2 signatures out of the buyer, seller and mediator.
  3. Buyer sends funds to this address.
  4. Seller ships the product.
  5. When the product arrives, buyer and seller sign a tx sending funds to seller.
  7. If the product doesn't arrive, mediator verifies this fact, and buyer and mediator sign a tx sending funds to buyer.
  8. If the product arrives but buyer refuses to pay, mediator verifies this fact, and seller and mediator sign a tx sending funds to seller.



Multi-signature transactions

- The general form of a multisig (*"M-of-N"*) transaction script looks like:
  - **"M <Public Key 1> <Public Key 2> … <Public Key N> N OP_CHECKMULTISIG"**
- In the case of a "2-of-3" multisig transaction the script looks like:
  - **"2 <Public Key A> <Public Key B> <Public Key C> 3 OP_CHECKMULTISIG"**
- The above forms a "locking script", which can only be unlocked by an equivalent "unlocking script", which contains 2 or more signatures computed from the signer's private keys and corresponding to the listed public keys, as follows:
  - **"OP_0 <Signature B> <Signature C>"**
- The above "locking" and "unlocking" script, together form a "validation script" which serves to enable multisig wallets:
  - **"OP_0 <Signature B> <Signature C>**
  - **2 <Public Key A> <Public Key B> <Public Key C> 3 OP_CHECKMULTISIG"**

(Validation script)

Exploring Blocks (CLI)

- Commands to use:

Command	Parameters	Description
getblock	<hash>	Returns information about the block with the given hash.
getblockhash	<index>	Returns hash of block in best-block-chain at <index>; index 0 is the genesis block

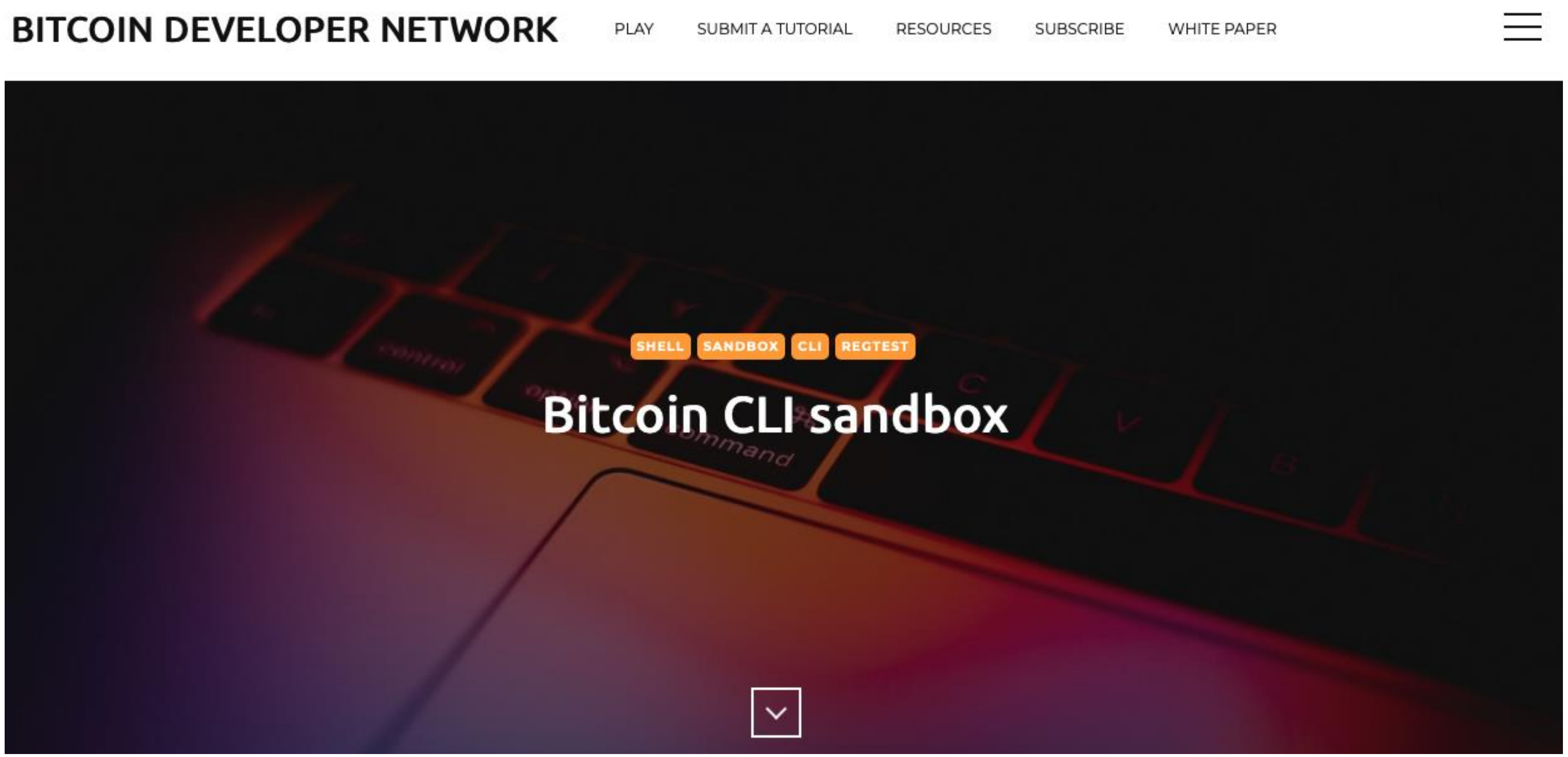
Exploring Blocks (CLI)

```
getblockhash 0
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

getblock 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
{
"hash" :
"000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
"confirmations" : 300344,
"size" : 285,
"height" : 0,
"version" : 1,
"merkleroot" :
"4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b",
"tx" : [
"4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
],
"time" : 1231006505,
"nonce" : 2083236893,
"bits" : "1d00ffff",
"difficulty" : 1.00000000,
"chainwork" :
"0000000000000000000000000000000000000000000000000000000100010001",
"nextblockhash" :
"00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048"
}
```

- We can retrieve block by index (“block height”), where “0” is the height of the *Genesis* block.
- The Genesis block only contains one transaction (the *coinbase* transaction). Prior to the first Bitcoin transaction (found on block 170), all blocks only contained the coinbase transaction.
- This is how bitcoins are being generated.

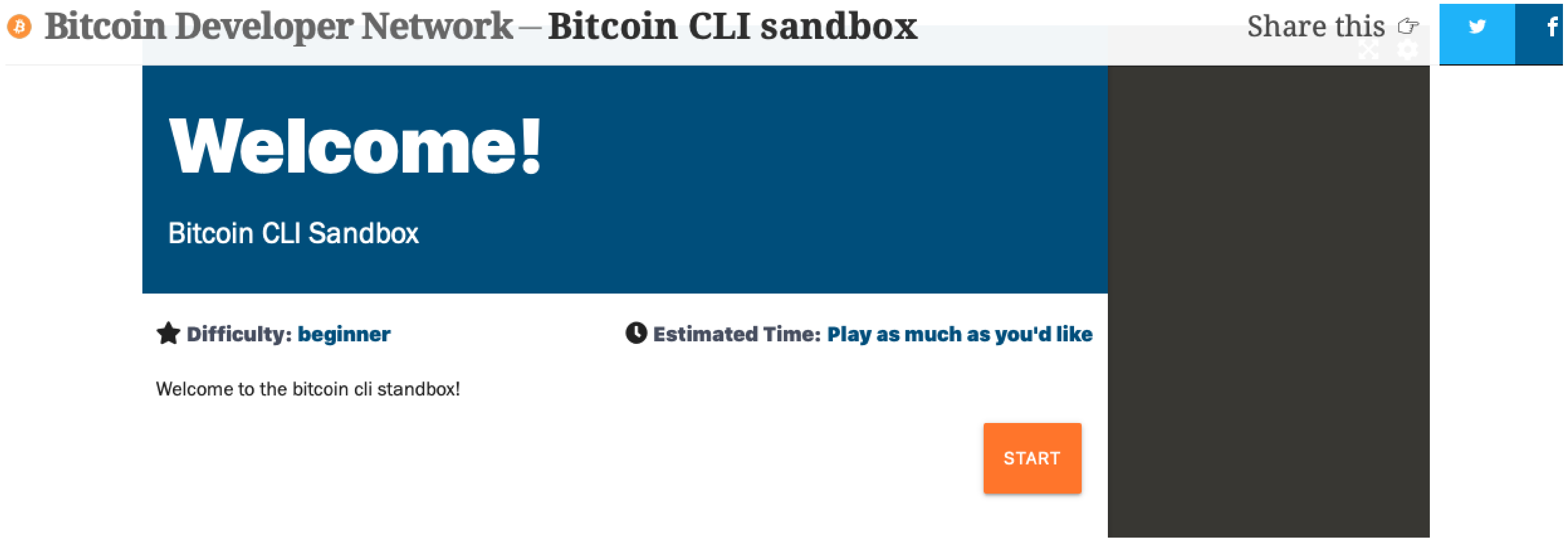
Sandbox

- https://bitcoindev.network/bitcoin-cli-sandbox/



Sandbox

- Move downwards and click “START"



Bitcoin Developer Network – Bitcoin CLI sandbox

Share this

Welcome!

Bitcoin CLI Sandbox

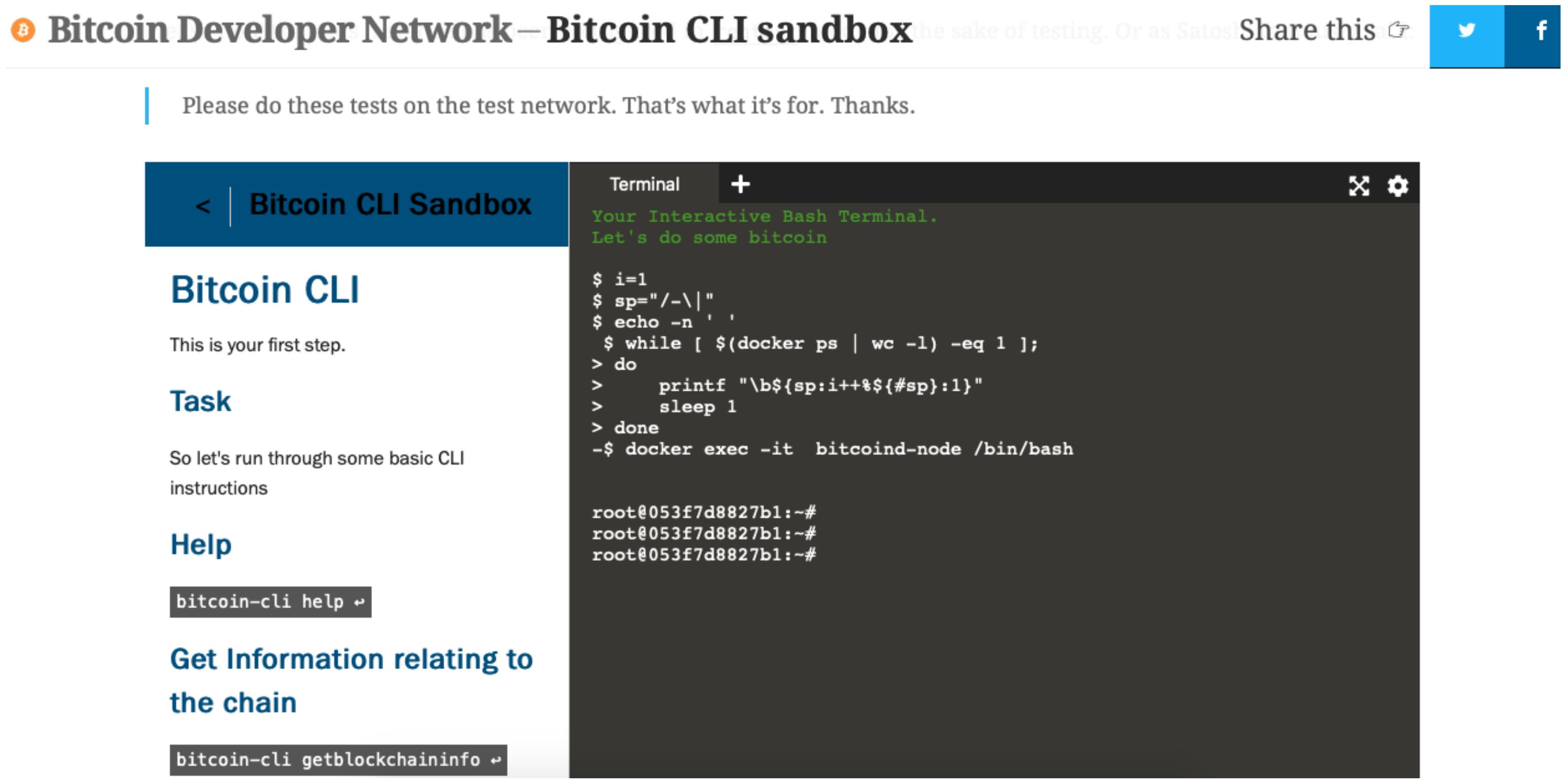
Difficulty: beginner

Estimated Time: Play as much as you'd like

Welcome to the bitcoin cli standbox!

Sandbox

- You can interact with the terminal (also, basic example in the left)



Bitcoin Developer Network – Bitcoin CLI sandbox

Share this

Please do these tests on the test network. That's what it's for. Thanks.

< | Bitcoin CLI Sandbox

Bitcoin CLI

This is your first step.

Task

So let's run through some basic CLI instructions

Help

`bitcoin-cli help ↵`

Get Information relating to the chain

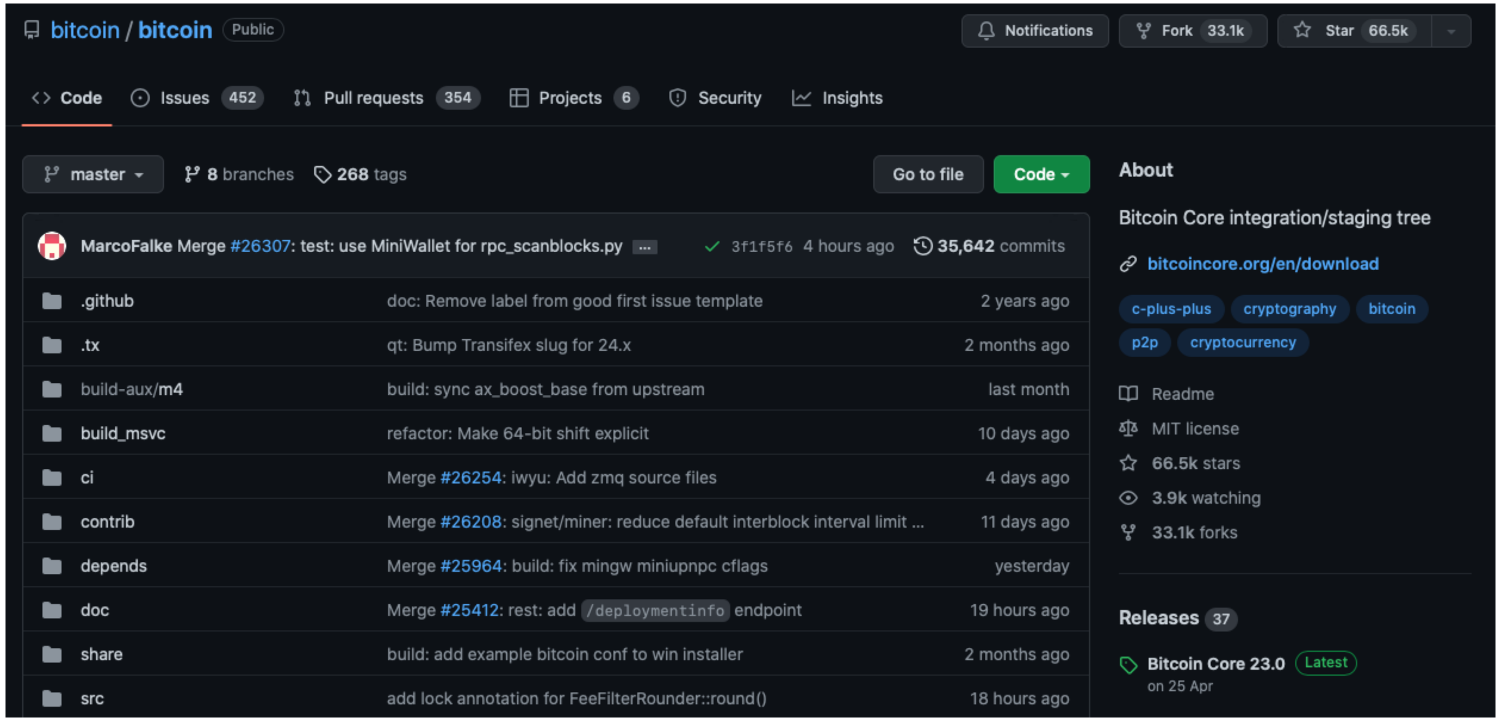
`bitcoin-cli getblockchaininfo ↵`

Terminal

```
Your Interactive Bash Terminal.
Let's do some bitcoin

$ i=1
$ sp="/-\|"
$ echo -n ' '
 $ while [ $(docker ps | wc -l) -eq 1 ];
> do
>     printf "\b${sp:i++%${#sp}:1}"
>     sleep 1
> done
-$ docker exec -it  bitcoind-node /bin/bash


root@053f7d8827b1:~#
root@053f7d8827b1:~#
root@053f7d8827b1:~#
```

Bitcoin Core: inspecting the source

- Bitcoin source code repository: https://github.com/bitcoin/bitcoin


bitcoin / bitcoin Public

Notifications | Fork 33.1k | Star 66.5k

Code | Issues 452 | Pull requests 354 | Projects 6 | Security | Insights

master | 8 branches | 268 tags | Go to file | Code

MarcoFalke Merge #26307: test: use MiniWallet for rpc_scanblocks.py ... ✓ 3f1f5f6 4 hours ago 35,642 commits

.github	doc: Remove label from good first issue template	2 years ago
.tx	qt: Bump Transifex slug for 24.x	2 months ago
build-aux/m4	build: sync ax_boost_base from upstream	last month
build_msvc	refactor: Make 64-bit shift explicit	10 days ago
ci	Merge #26254: iwyu: Add zmq source files	4 days ago
contrib	Merge #26208: signet/miner: reduce default interblock interval limit ...	11 days ago
depends	Merge #25964: build: fix mingw miniupnpc cflags	yesterday
doc	Merge #25412: rest: add `/deploymentinfo` endpoint	19 hours ago
share	build: add example bitcoin conf to win installer	2 months ago
src	add lock annotation for FeeFilterRounder::round()	18 hours ago

About

Bitcoin Core integration/staging tree

bitcoincore.org/en/download

c-plus-plus cryptography bitcoin p2p cryptocurrency

Readme
MIT license
66.5k stars
3.9k watching
33.1k forks

Releases 37

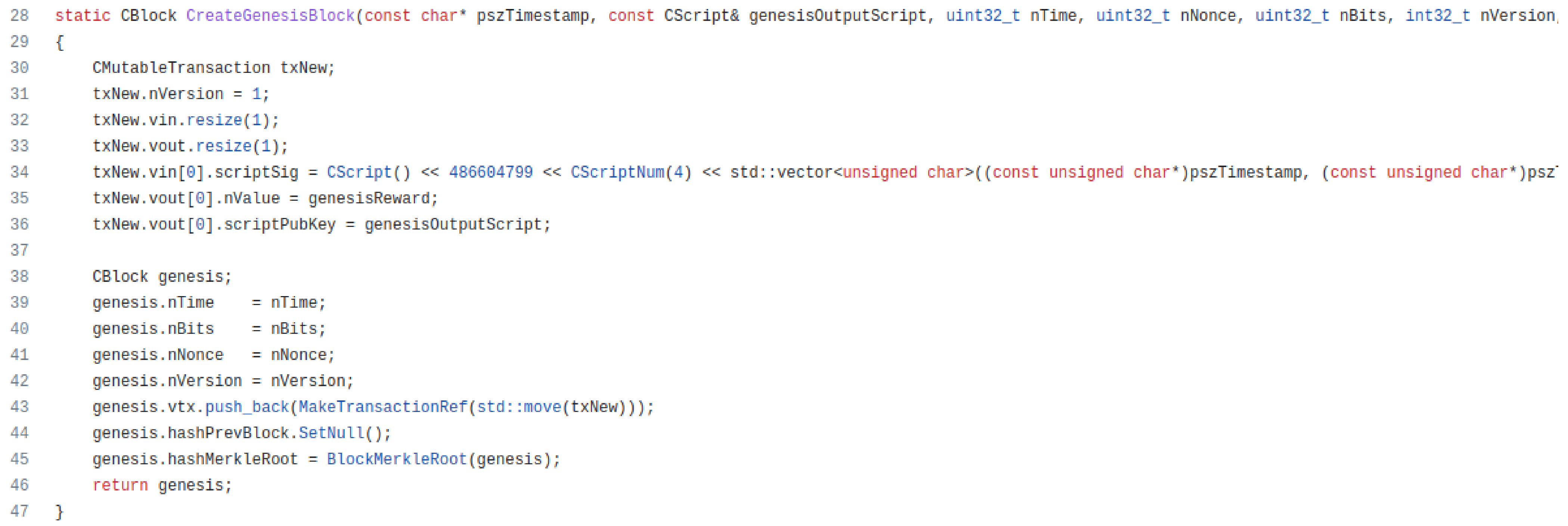
Bitcoin Core 23.0 Latest
on 25 Apr

- Even with zero programming background, the inspection (i.e., reading) of source code [*] is interesting, informative and in many cases eye opening. [*] see “src” sub-folder

Bitcoin Core: inspecting the source

Example: bitcoin/src/kernel/chainparams.cpp

- Creation of the genesis block



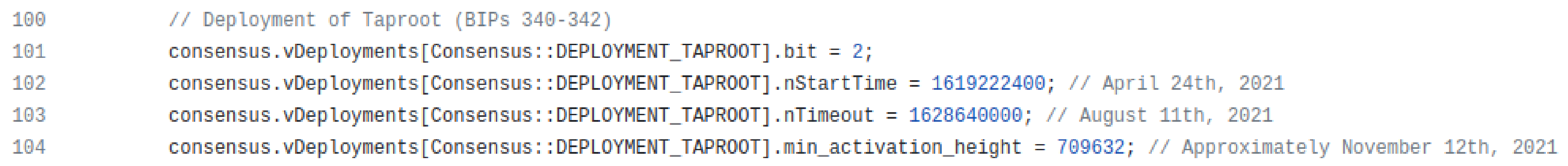
```
static CBlock CreateGenesisBlock(const char* pszTimestamp, const CScript& genesisOutputScript, uint32_t nTime, uint32_t nNonce, uint32_t nBits, int32_t nVersion,
{
    CMutableTransaction txNew;
    txNew.nVersion = 1;
    txNew.vin.resize(1);
    txNew.vout.resize(1);
    txNew.vin[0].scriptSig = CScript() << 486604799 << CScriptNum(4) << std::vector<unsigned char>((const unsigned char*)pszTimestamp, (const unsigned char*)psz
    txNew.vout[0].nValue = genesisReward;
    txNew.vout[0].scriptPubKey = genesisOutputScript;

    CBlock genesis;
    genesis.nTime    = nTime;
    genesis.nBits    = nBits;
    genesis.nNonce   = nNonce;
    genesis.nVersion = nVersion;
    genesis.vtx.push_back(MakeTransactionRef(std::move(txNew)));
    genesis.hashPrevBlock.SetNull();
    genesis.hashMerkleRoot = BlockMerkleRoot(genesis);
    return genesis;
}
```

- Interesting to observe the list of parameters

Bitcoin Core: inspecting the source

Example: bitcoin/src/kernel/chainparams.cpp

- About the Taproot update



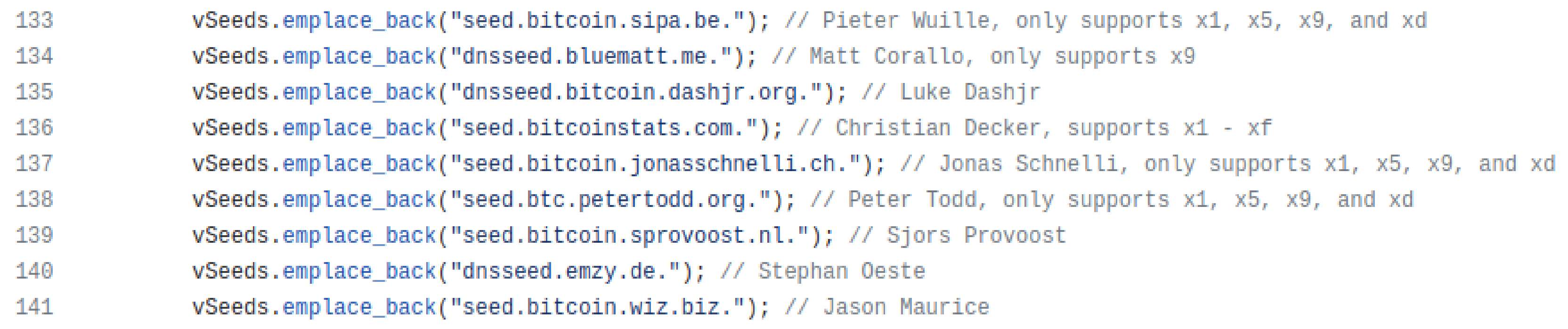
```
100         // Deployment of Taproot (BIPs 340-342)
101         consensus.vDeployments[Consensus::DEPLOYMENT_TAPROOT].bit = 2;
102         consensus.vDeployments[Consensus::DEPLOYMENT_TAPROOT].nStartTime = 1619222400; // April 24th, 2021
103         consensus.vDeployments[Consensus::DEPLOYMENT_TAPROOT].nTimeout = 1628640000; // August 11th, 2021
104         consensus.vDeployments[Consensus::DEPLOYMENT_TAPROOT].min_activation_height = 709632; // Approximately November 12th, 2021
```

- Notice how time is measured (e.g., see lines 102 and 103)
- Some values are hard-coded
    - Discussion: Any comments?

Bitcoin Core: inspecting the source

Example: bitcoin/src/kernel/chainparams.cpp

- Collection of “seeds”

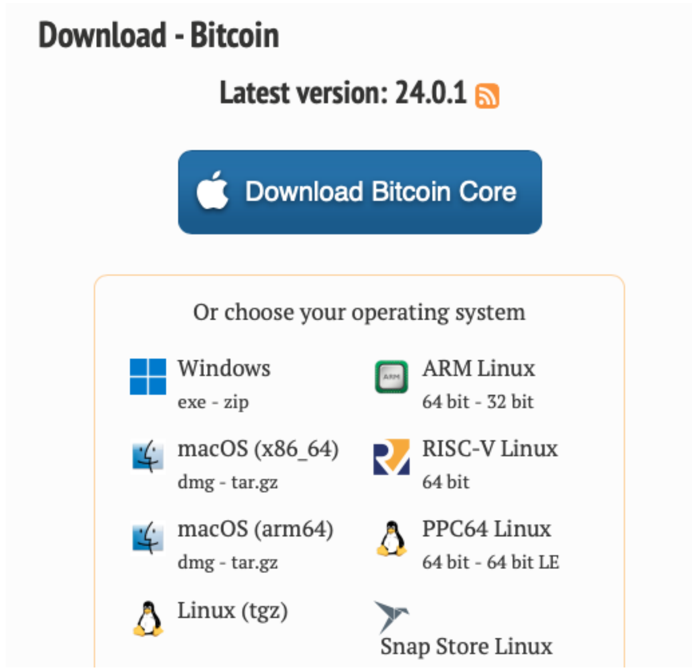


```
133        vSeeds.emplace_back("seed.bitcoin.sipa.be."); // Pieter Wuille, only supports x1, x5, x9, and xd
134        vSeeds.emplace_back("dnsseed.bluematt.me."); // Matt Corallo, only supports x9
135        vSeeds.emplace_back("dnsseed.bitcoin.dashjr.org."); // Luke Dashjr
136        vSeeds.emplace_back("seed.bitcoinstats.com."); // Christian Decker, supports x1 - xf
137        vSeeds.emplace_back("seed.bitcoin.jonasschnelli.ch."); // Jonas Schnelli, only supports x1, x5, x9, and xd
138        vSeeds.emplace_back("seed.btc.petertodd.org."); // Peter Todd, only supports x1, x5, x9, and xd
139        vSeeds.emplace_back("seed.bitcoin.sprovoost.nl."); // Sjors Provoost
140        vSeeds.emplace_back("dnsseed.emzy.de."); // Stephan Oeste
141        vSeeds.emplace_back("seed.bitcoin.wiz.biz."); // Jason Maurice
```

- Again, the (short) list of seeds is hard-coded (line 133-141)
  - Discussion: Any comments wrt decentralization and security?

Latest Bitcoin Core Release (Dec 2022)

- Installing Bitcoin Core
- How to set up the Bitcoin Core wallet Client for Beginners and send your first Transaction
- Bitcoin Core
- Latest Bitcoin Core release (Dec. 12, 2022): 24.0.1



Latest Bitcoin Core Release (Dec 2022)

- Overview of update types:
  - P2P and network changes
  - Updated RPCs
  - New RPCs
  - Updated REST APIs
  - Wallet
    - Notable feature: Miniscript

- More information:
  - https://bitcoincore.org/en/releases/24.0.1/
  - https://bitcoinmagazine.com/technical/bitcoin-core-24-0-released-what-is-new

Latest Bitcoin Core Release (Dec 2022): Miniscript

- A purpose-specific language for developing a subset of Bitcoin scripts

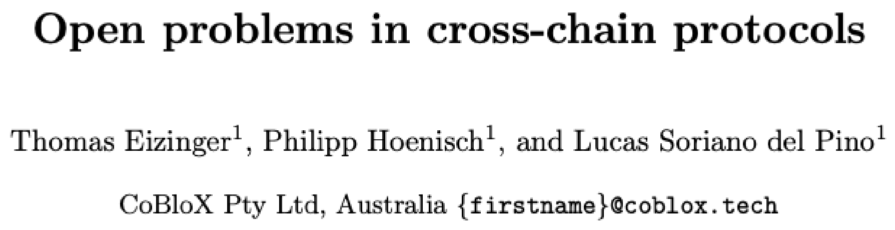
- Main characteristic: development of spending conditions relying on combinations of:
  - Signatures
  - Hash locks
  - Time locks

- On this basis, the following (special) functionality is supported:
  - Determine the optimal script that addresses a specific set of spending conditions
  - Support multisig scripts that include other mutlisig transactions
  - Identify the spending conditions of a script
  - Given a script, be able to predict the cost of spending an output.
  - Estimation of cost with regards to outputs spending
  - Estimation of opcodes-related limitations

Latest Bitcoin Core Release (Dec 2022): Miniscript

Status in 2021

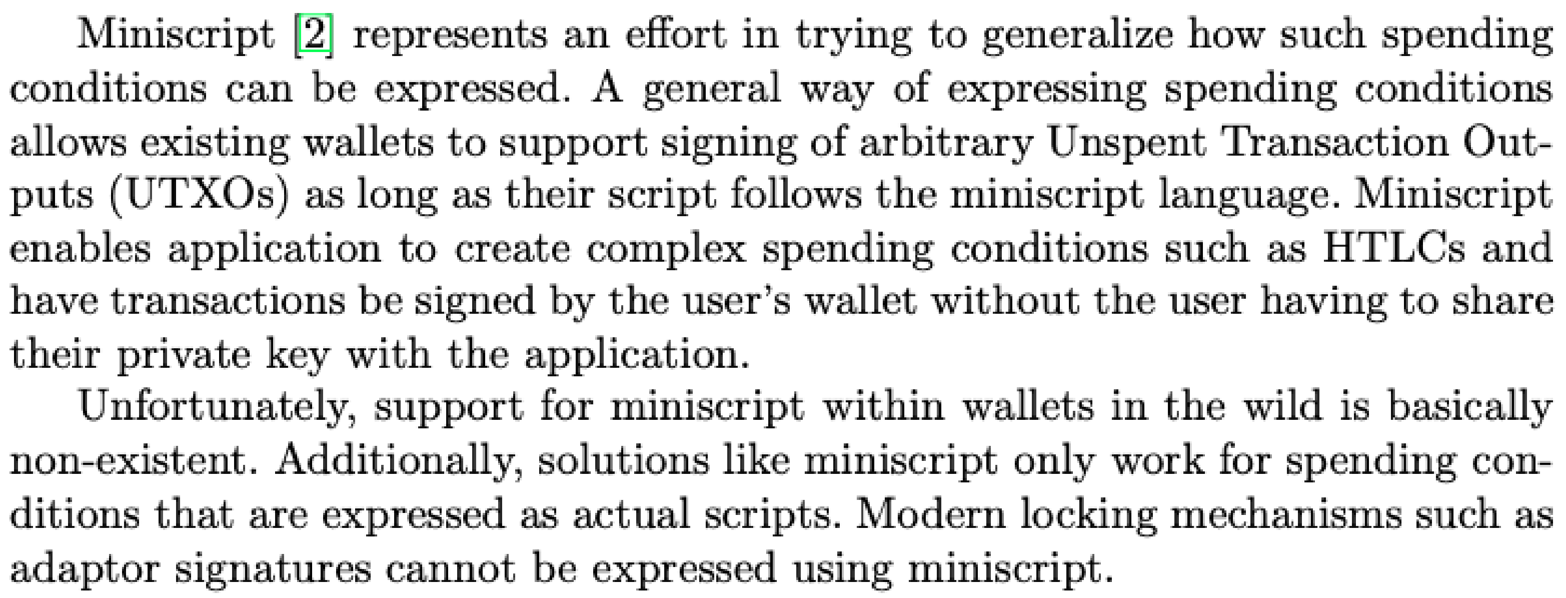
- https://arxiv.org/pdf/2101.12412.pdf



Open problems in cross-chain protocols

Thomas Eizinger[1], Philipp Hoenisch[1], and Lucas Soriano del Pino[1]

CoBloX Pty Ltd, Australia {firstname}@coblox.tech



Miniscript [2] represents an effort in trying to generalize how such spending conditions can be expressed. A general way of expressing spending conditions allows existing wallets to support signing of arbitrary Unspent Transaction Outputs (UTXOs) as long as their script follows the miniscript language. Miniscript enables application to create complex spending conditions such as HTLCs and have transactions be signed by the user's wallet without the user having to share their private key with the application.

Unfortunately, support for miniscript within wallets in the wild is basically non-existent. Additionally, solutions like miniscript only work for spending conditions that are expressed as actual scripts. Modern locking mechanisms such as adaptor signatures cannot be expressed using miniscript.

Conclusions

- Studying the basic principles of the Bitcoin Scripting Language (how to read a script etc) and the basic principles of the Bitcoin Core:
    - We should be able to monitor the basic issues of the Bitcoin
    - Code inspection with zero programming background is possible (and it should be attempted)
        - More on this very soon
- The issue is not to focus only to the technical parts but also:
    - To capitalize our knowledge from the basic technical principles
    - To be able to follow all the developments by
        - Understanding their functionality
        - Understanding their significance
- Keep an eye on updates
    - New and updated functionality
    - Deprecated functionality

Bibliography

References

- Andreas M. Antonopoulos (2017). *"Mastering Bitcoin: Programming the Open Blockchain".* 2nd Edition. Sebastopol: O'Reilly Media
  - Chapter 3
- Optional additional references: research article (especially Section 1 and 7)
  - Gennaro, R., Goldfeder, S., and Narayanan, A. (2016). *"Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security".* In Proc. of International Conference on Applied Cryptography and Network Security (pp. 156-174). Springer, Cham. [available here]

  The authors propose a threshold DSA algorithm for distributing the signing power among multiple parties. The algorithm is applied for the use if Bitcoin wallets.

UNIVERSITY *of* NICOSIA

Instructor’s Email:

iosif.e@unic.ac.cy