# Bitcoin address

An identifier of 26-35 alphanumeric characters, beginning with the number 1, 3 or bc1 (m, n or 2 for testnet) that represents a possible destination for a bitcoin payment.

Uppercase letter "O", uppercase letter "I", lowercase letter "l", and the number "0" are never used to prevent visual ambiguity

Addresses can be generated at no cost by any user of Bitcoin.

There are currently three address formats in use:

- P2PKH (public key hash), starts with 1: 1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2.
- P2SH (script hash, multisig) starts with 3, eg: 3J98t1WpEZ73CNmQviecrnyiWrnqRhWNLy.
- Bech32 type starting with bc1, eg: bc1qar0srrr7xfkvy5l643lydnw9re59gtzzwf5mdq.

Its a single use token.

Can be created offline.

P2PKH and P2SH are case sensitive, Bech32 is case insensitive.

Protected by CRC: the probability that a mistyped address is accepted as being valid is 1 in $2^{32}$, that is, approximately 1 in 4.29 billion.

See details: https://en.bitcoin.it/wiki/Address

# Bitcoin address «balance» or «from» address

Addresses are not wallets nor accounts, and do not carry balances. There is no «address balance» or «wallet address» in Bitcoin.

They only receive funds, and you do not send "from" an address at any time.

Bitcoin transactions do not have any kind of origin-, source- or "from" address.

When you transfer bitcoin from address you usually take all coins from it, transfer needed amount and return the rest to «change» new address in your wallet.

See details: https://en.bitcoin.it/wiki/Address

# Avoid Bitcoin address reuse

It is not a «bank account number» but «invoice». **Do not reuse!**

Protects privacy and keeps Bitcoins secure from hypothetical attacks by quantum computers.

But you can! For example:
https://www.blockchain.com/en/btc/tx/8d6e61cbdd67cc368882d2d5b8d79d9c542d4190ba96144bf287e618e8e27123

Address reuse refers to the use of the same address for multiple transactions. It is an unintended practice, abusing the privacy and security of the participants of the transactions as well as future holders of their value. It also only functions by accident, not by design, so cannot be depended on to work reliably.

Worked Example 1 - Savings Revealed

You save in bitcoin, using a single-address paper wallet.

All your bitcoin savings to this same address, let's say it contains $1 million worth.

You buy a small amount of bitcoins to add to your savings, depositing in the paper wallet.

The person who sold you the bitcoins follows their trail on the blockchain and finds your paper wallet containing $1 million.

He mentions it to someone in a cafe or bar.

Word gets around. A burglar raids your home. Kidnappers capture your children and know exactly how much to demand in ransom.

Other motivation examples:
https://en.bitcoin.it/wiki/Address_reuse

# Bitcoin v1 address technical details

A Bitcoin address is a 160-bit hash of the public portion of a public/private ECDSA keypair. Private key is a random number. Public key is a curve point (x,y).

0 - Having a private ECDSA key
18e14a7b6a307f426a94f8114701e7c8e774e7f9a47e2c2035db29a206321725

1 - Take the corresponding public key generated with it (33 bytes, 1 byte 0x02 (y-coord is even), and 32 bytes corresponding to X coordinate)
0250863ad64a87ae8a2fe83c1af1a8403cb53f53e486d8511dad8a04887e5b2352

2 - Perform SHA-256 hashing on the public key
0b7c28c9b7290c98d7438e70b3d3f7c848fbd7d1dc194ff83f4f7cc9b1378e98

3 - Perform RIPEMD-160 hashing on the result of SHA-256
f54a5851e9372b87810a8e60cdd2e7cfd80b6e31

4 - Add version byte in front of RIPEMD-160 hash (0x00 for Main Network)
00f54a5851e9372b87810a8e60cdd2e7cfd80b6e31

5 - Perform SHA-256 hash on the extended RIPEMD-160 result
ad3c854da227c7e99c4abfad4ea41d71311160df2e415e713318c70d67c6b41c

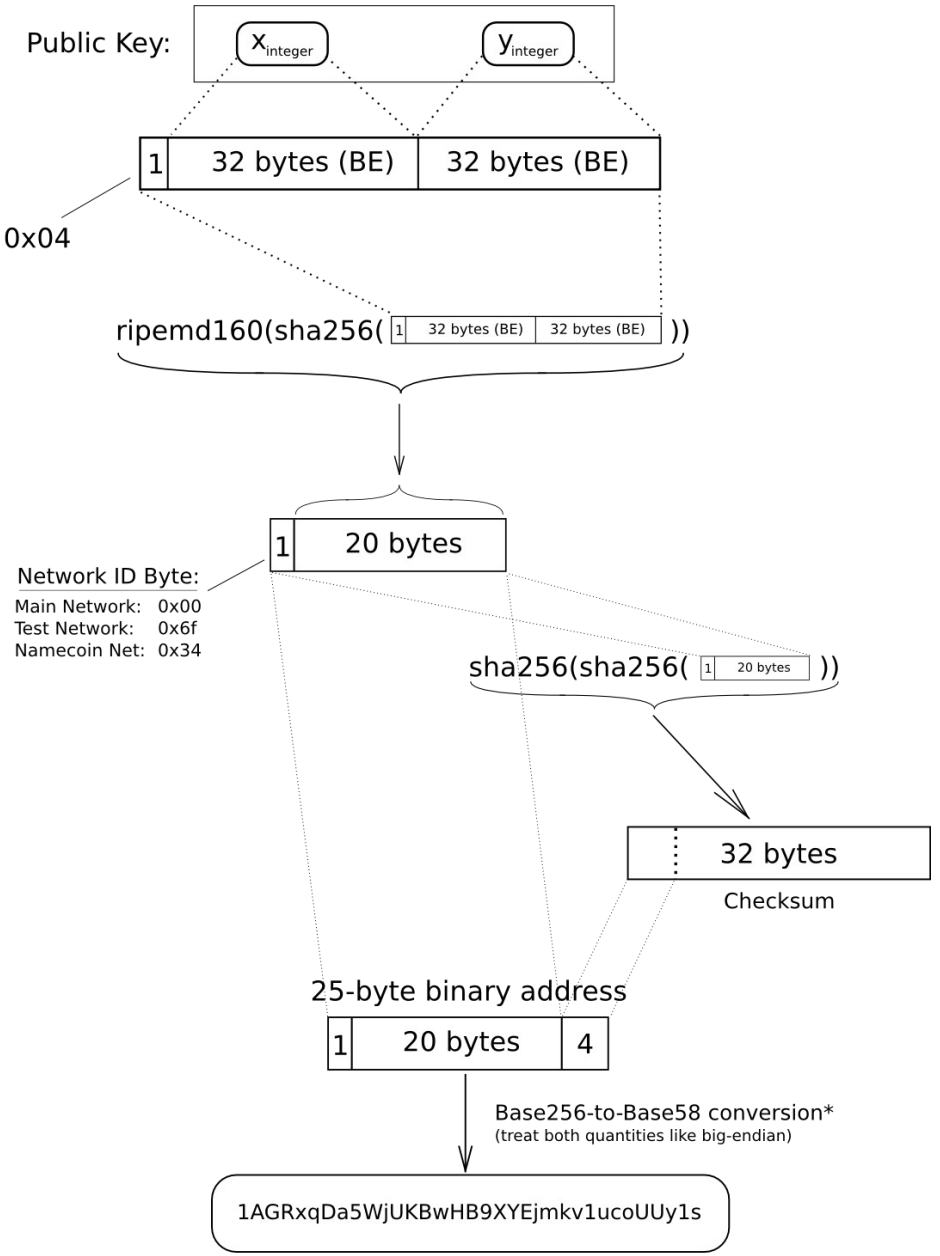
6 - Perform SHA-256 hash on the result of the previous SHA-256 hash
c7f18fe8fcbed6396741e58ad259b5cb16b7fd7f041904147ba1dcffabf747fd

7 - Take the first 4 bytes of the second SHA-256 hash. This is the address checksum
c7f18fe8

8 - Add the 4 checksum bytes from stage 7 at the end of extended RIPEMD-160 hash from stage 4. This is the 25-byte binary Bitcoin Address.
00f54a5851e9372b87810a8e60cdd2e7cfd80b6e31c7f18fe8

9 - Convert the result from a byte string into a base58 string using Base58Check encoding.
1PMycacnJaSqwwJqjawXBErnLsZ7RkXUAs

# Elliptic-Curve Public Key to BTC Address conversion




*In a standard base conversion, the 0x00 byte on the left would be irrelevant (like writing '052' instead of just '52'), but in the BTC network the left-most zero chars are carried through the conversion. So for every 0x00 byte on the left end of the binary address, we will attach one '1' character to the Base58 address. This is why main-network addresses all start with '1'

# Bitcoin wallet

Wallet is a **set of private keys** and a set of addresses generated with it.

A paper wallet is the name given to an obsolete and unsafe method of storing bitcoin which was popular between 2011 and 2016.
It works by having a single private key and bitcoin address, usually generated by a website, being printed out onto paper, promotes address reuse.

Modern HD wallets (BIP0032) use Seed Phrases to store private keys. It is 12 random english words from 2048 word dictionary (132 bits of security, 2^132 combinations). Hierarchical Deterministic wallet is a system of deriving keys from a single starting point known as a seed. So you only store one private key and all public keys for addresses can be recovered from it.

Additional password protection ($13^{th}$ word), master password to encrypt wallet on a disk. You need to “have something” and “know something” to recover all Bitcoin on the blockchain!

How to store bitcoins safe:
- bitcoin wallets should be backed up by writing down their seed phrase (stores private key),
- this phrase must be kept safe and secret (pencil and paper, engraved into metal!),
- when sending or receiving transactions the wallet software should obtain information about the bitcoin network from your own full node.

# SegWit

In 2017 BIP-0173 soft fork was activated with Bech32 addresses (SegWit address or P2WPKH). Starting from bc1 (tb1 in testnet)

SegWit = Segregated Witness

P2WPKH = Pay to Witness Public Key Hash

It removes permission scripts from transactions, storing them off the blockchain and only storing hash of scripts in blocks.

It increased number of transactions per block and decreased mining fees.

Makes other data anchoring in Bitcoin blockchain possible, like LightCoin or smart contracts.

Bech32 is not accepted by all nodes and clients.

Recent Copay uses it by default. Can be disabled in Copay.

https://www.blockchain.com/btc-testnet/tx/eab998c93bbbacc988a6b02931304305b2aae4ff753ef558043a92fa8aba7684

Blockchain.com Wallet Exchange Explorer Buy Bitcoin Trade

Testnet is an alternative Bitcoin blockchain that developers use for testing. Testnet coins do not hold any value.

Explorer > Bitcoin Testnet > Transaction USD Search your transaction, an address or a block

## Summary

USD BTC

| | | |
|---|---|---|
| Fee | 0.00000141 BTC<br>(0.632 sat/B - 0.251 sat/WU - 223 bytes)<br>(1.000 sat/vByte - 141 virtual bytes) | 0.02276328 BTC |
| Hash | eab998c93bbbacc988a6b02931304305b2aae4ff753ef558043a92f... | 2021-10-22 19:23 |

| | | | |
|---|---|---|---|
| tb1qf2kaqsscfqdvmguh846jzxvdw2006gpvsyejpx | 0.02276469 BTC | tb1qttsmxju7qhjs33mfxmnysnhjvy5me449a4elsv | 0.01276328 BTC |
| | | tb1qt553kqtvwzwwq3fu7xkassurh46jj3zc742dmx | 0.01000000 BTC |

This transaction was first broadcast to the Bitcoin network on October 22, 2021 at 7:23 PM GMT+3. The transaction currently has 3,512 confirmations on the network. At the time of this transaction, 0.02276328 BTC was sent with a value of $1,392.88. The current value of this transaction is now $1,446.58. Learn more about how transactions work.

## Details

| | |
|---|---|
| Hash | eab998c93bbbacc988a6b02931304305b2aae4ff753ef558043a92fa8aba7684 |
| Status | Confirmed |
| Received Time | 2021-10-22 19:23 |
| Size | 223 bytes |

# Bitcoin addresses in code

| Decimal prefix | Hex | Example use | Leading symbol(s) | Example |
|---|---|---|---|---|
| 0 | 00 | Pubkey hash (P2PKH address) | 1 | 17VZNX1SN5NtKa8UQFxwQbFeFc3iqRYhem |
| 5 | 05 | Script hash (P2SH address) | 3 | 3EktnHQD7RiAE6uzMj2ZifT9YgRrkSgzQX |
| 128 | 80 | Private key (WIF, uncompressed pubkey) | 5 | 5Hwgr3u458GLafKBgxtssHSPqJnYoGrSzgQsPwLFhLNYskDPyyA |
| 128 | 80 | Private key (WIF, compressed pubkey) | K *or* L | L1aW4aubDFB7yfras2S1mN3bqg9nwySY8nkoLmJebSLD5BWv3ENZ |
| 4 136 178 30 | 0488B21E | BIP32 pubkey | xpub | xpub661MyMwAqRbcEYS8w7XLSVeEsBXy79zSzH1J8vCdxAZningWLdN3zgtU6LBpB85b3D2yc8sfvZU521AAwdZafEz7mnzBBsz4wKY5e4cp9LB |
| 4 136 173 228 | 0488ADE4 | BIP32 private key | xprv | xprv9s21ZrQH143K24Mfq5zL5MhWK9hUhhGbd45hLXo2Pq2oqzMMo63oStZzF93Y5wvzdUayhgkkFoicQZcP3y52uPPxFnfoLZB21Teqt1VvEHx |
| 111 | 6F | Testnet pubkey hash | m *or* n | mipcBbFg9gMiCh81Kj8tqqdgoZub1ZJRfn |
| 196 | C4 | Testnet script hash | 2 | 2MzQwSSnBHWHqSAqtTVQ6v47XtaisrJa1Vc |
| 239 | EF | Testnet Private key (WIF, uncompressed pubkey) | 9 | 92Pg46rUhgTT7romnV7iGW6W1gbGdeezqdbJCzShkCsYNzyyNcc |
| 239 | EF | Testnet Private key (WIF, compressed pubkey) | c | cNJFgo1driFnPcBdBX8BrJrpxchBWXwXCvNH5SoSkdcF6JXXwHMm |
| 4 53 135 207 | 043587CF | Testnet BIP32 pubkey | tpub | tpubD6NzVbkrYhZ4WLczPJWReQycCJdd6YVWXubbVUFnJ5KgU5MDQrD998ZJLNGbhd2pq7ZtDiPYTfJ7iBenLVQpYgSQqPjUsQeJXH8VQ8xA67D |
| 4 53 131 148 | 04358394 | Testnet BIP32 private key | tprv | tprv8ZgxMBicQKsPcsbCVeqqF1KVdH7gwDJbxbzpCxDUsoXHdb6SnTPYxdwSAKDC6KKJzv7khnNWRAJQsRA8BBQyiSfYnRt6zuu4vZQGKjeW4YF |
| | | Bech32 pubkey hash or script hash | bc1 | bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4 |
| | | Bech32 testnet pubkey hash or script hash | tb1 | tb1qw508d6qejxtdg4y5r3zarvary0c5xw7kxpjzsx |

To understand serialization formats see https://en.bitcoin.it/wiki/List_of_address_prefixes

# Bitcoinjs Node.JS Bitcoin Cryptography library

Allows to programmatically generate keys, transform serialization formats, import/export wallets, craft and broadcast transactions programmatically:

https://github.com/bitcoinjs/bitcoinjs-lib

Examples in this lecture are for version 4. Can be used with Bitcoin v1 addresses with SegWit disabled in Copay.

Txbt example is for version 5.2 (SegWit).

6+ for TypeScript.

# Create address

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

const TESTNET = bitcoin.networks.testnet

const keyPair = bitcoin.ECPair.makeRandom(TESTNET)
const { address } = bitcoin.payments.p2pkh({ pubkey: keyPair.publicKey,
network: TESTNET })

console.log(address)

// Bech32 address
const { address } = bitcoin.payments.p2wpkh({ pubkey:
keyPair.publicKey, network: TESTNET })

console.log(address)
```

# Save address to WIF

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

const TESTNET = bitcoin.networks.testnet

const keyPair = bitcoin.ECPair.makeRandom(TESTNET)
const { address } = bitcoin.payments.p2pkh({ pubkey: keyPair.publicKey,
network: TESTNET })

const WIF = keyPair.toWIF()

console.log(WIF)
```

# Load address from WIF

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

const TESTNET = bitcoin.networks.testnet

const WIF = 'string_from_previous_script_here'
const keyPair = bitcoin.ECPair.fromWIF(WIF)
keyPair.network = TESTNET // hack to fix bitcoinjs bug
const { address } = bitcoin.payments.p2pkh({ pubkey: keyPair.publicKey,
network: bitcoin.networks.testnet})

console.log(address)
```

# Create transaction «Hello World»

```
var tx = new bitcoin.TransactionBuilder(bitcoin.networks.testnet)

// Add the input (who is paying):
// [previous transaction hash, index of the output to use]
var txId = 'aa94ab02c182214f090e99a0d57021caffd0f195a81c24602b1028b130b63e31'
tx.addInput(txId, 0)

// Add the output (who to pay to):
// [payee's address, amount in satoshis, 1 bitcoin =  100 000 000 satoshis]
tx.addOutput("1Gokm82v6DmtwKEB8AiVhm82hyFSsEvBDK", 15000)

// Initialize a private key using WIF
var privateKeyWIF = 'L1uyy5qTuGrVXrmrsvHWHgVzW9kKdrp27wBC7Vs6nZDTF2BRUVwy'
var keyPair = bitcoin.ECPair.fromWIF(privateKeyWIF)

// Sign the first input with the new key
tx.sign(0, keyPair)

// Print transaction serialized as hex
console.log(tx.build().toHex())
```

TODO: add second output using correct transaction from testnet as an example, the miner bonus = input – sum of all outputs, should be > 0

# Tricky Part connecting with Copay

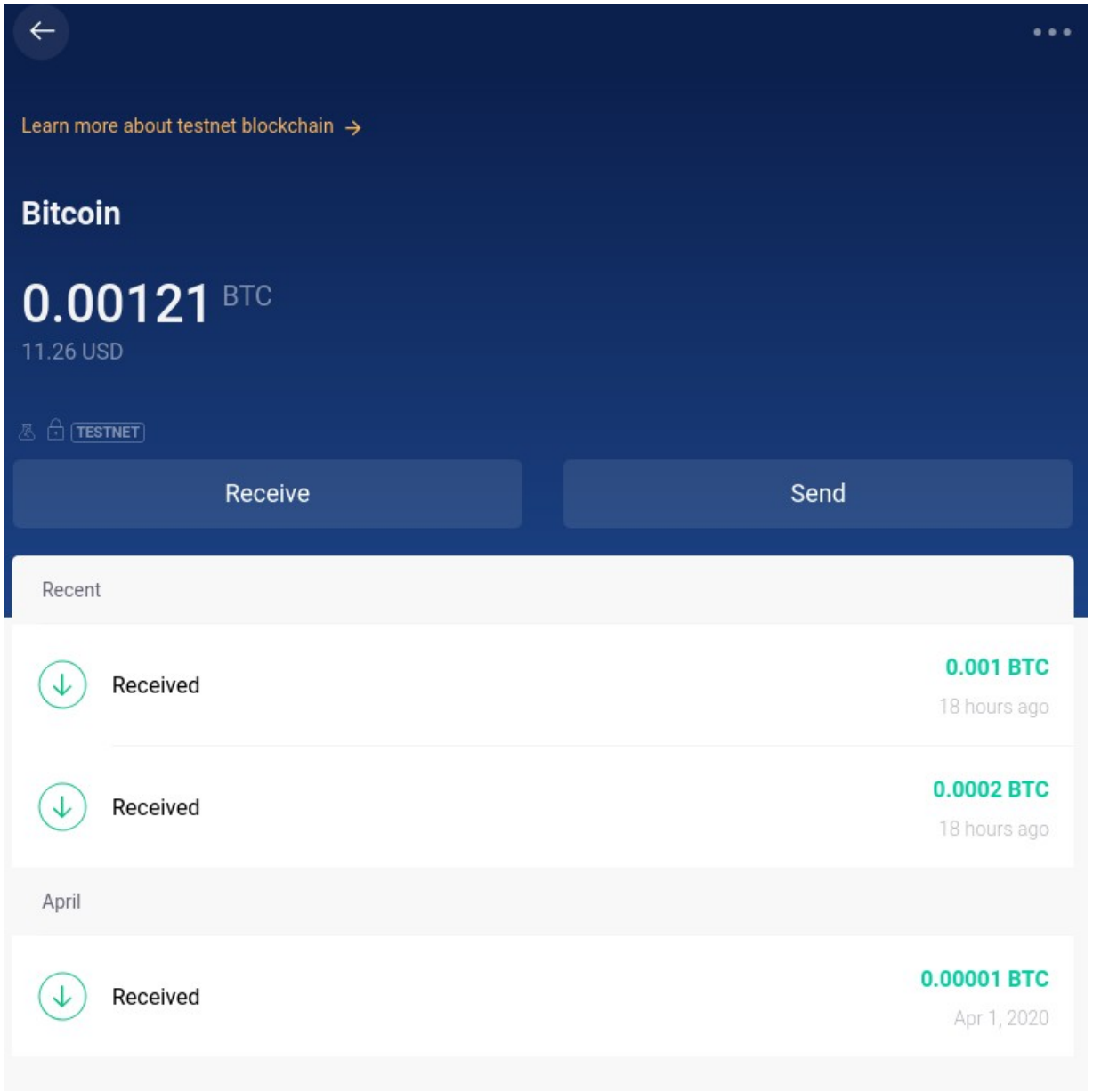
```
// Get WIF from extended private key exported from Copay (sender)
const t =
bip32.fromBase58('tprv8ZgxMBicQKsPexegQB353wsJRdnYgj4CimrYLaSiGoEJQ39aTa464L
5czHLFyCrfEf3Wys6ZNhy3xLEVCyL33JJ7ZRm2ym7bSHjFtCxsq9T', TESTNET);

// BIP44 path, bitcoin, account 0, external address
// See https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki
// m / purpose' / coin_type' / account' / change / address_index
// set change = 1 for change address
// address_index starts from 0 and up
const derivedIndex = "m/44'/1'/0'/0/0";
const wifKey = t.derivePath(derivedIndex).toWIF();
const keyPair2 = bitcoin.ECPair.fromWIF(wifKey, TESTNET); // Sign transaction with this key.
console.log('WIF_In=',keyPair2.toWIF())

// Get WIF from extended private key exported from Copay (receiver)
const c =
bip32.fromBase58('tprv8ZgxMBicQKsPd18wdhpYEXeMzwXPite9m3YPw84USg3Bu6K9i35x18
KXys25N914CrBpd2ScThXtd66EMucNxZsu6dCaRVyPf9BcboZoDhP', TESTNET)const
wifKey2 = c.derivePath(derivedIndex).toWIF();
const keyPair = bitcoin.ECPair.fromWIF(wifKey2, TESTNET);
console.log('WIF_out=',keyPair.toWIF());

// Get address
const {address} = bitcoin.payments.p2pkh({pubkey: keyPair.publicKey,
network: bitcoin.networks.testnet})
console.log('Destination=',address)
```

# Receive testnet Bitcoin in Copay



# Receive testnet Bitcoin in Copay

## Transaction

**Transaction Hash** a3c7bb8a657f7526b1be8d68ae6c50c4546df9446ce88cb9fa3b07736efe971b

## Summary

| | |
|---|---|
| **Size** | 140 (bytes) |
| **Fee Rate** | 1.2 sats/byte |
| **Received Time** | May 6, 2020 at 8:37:04 PM GMT+3 |
| **Included in Block** | 00000000000001150a9388206ab0cdc2cf6c45ebd9f8531af0cb1a70e2af70a |

## Details

a3c7bb8a657f7526b1be8d68ae6c50c4546df9446ce88cb9fa3b07736efe971b

| | | | |
|---|---|---|---|
| 2MvpzrKKSRrY4vagTCnLMNw12Djoa... | 0.13542403 BTC | mz4D1JP1F88M9qBuT4rYT2X4r8h3G52gR4 | 0.0002 BTC (U) |
| | | 2Msdc9xwAJwM1icDWVX2QYLpCP6RAsE5rSx | 0.13522235 BTC (S) |

FEE 0.00000168 BTC

76 CONFIRMATIONS

0.13542235 BTC

# Import Copay Key, fake testnet and spend Bitcoins programmatically

```
const bitcoin = require('bitcoinjs-lib')
const bip32 = require('bip32')

const TESTNET = bitcoin.networks.testnet

// Get WIF from mainnet extended private key exported from recent Copay (sender)
const t =
bip32.fromBase58('xprv9s21ZrQH143K3eNbD7b24NexqypNUq4v4GFFCgyQmobVqG3qp4sxRAAYtXXz8J77pv6izYd5Cu5qpFJMS6GbzGw7L4BYR83cFvBrMfVidMn');
t.network = TESTNET; // trick to transform mainnet key pair to testnet

// BIP44 (Hierarchical Deterministic Wallet) path, bitcoin testnet, account 0, external address
// See https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki
// m / purpose' / coin_type' / account' / change / address_index
// coin_type = 1 for testnet, 0 for mainnet
// set change = 1 for change address
// address_index starts from 0 and up
const derivedIndex = "m/44'/1'/0'/0/4"; // Tricky part is to calculate last parameter (4) to correspond to derived address.

const wifKey = t.derivePath(derivedIndex).toWIF();
const keyPair = bitcoin.ECPair.fromWIF(wifKey, TESTNET); // Sign transaction with this key.

const { address } = bitcoin.payments.p2pkh({ pubkey: keyPair.publicKey, network: TESTNET });
//console.log(address); // Check that address is correctly derrived and corresponds to what you see in Copay receiving transaction.

// Craft transaction.
var tx = new bitcoin.TransactionBuilder(TESTNET);

// Add the input (who is paying):
// [previous transaction hash, index of the output to use]
var txId = 'a3c7bb8a657f7526b1be8d68ae6c50c4546df9446ce88cb9fa3b07736efe971b'
tx.addInput(txId, 0);

// Add the output (who to pay to):
// [payee's address, amount in satoshis, 1 bitcoin =  100 000 000 satoshis]
tx.addOutput("morGGeujk1h822bAji3xmqJRjYNxCQqttg", 0.00019 * 100000000);

// Sign the first input with the imported key
tx.sign(0, keyPair)

// Print transaction serialized as hex for validation and posting
console.log(tx.build().toHex())
```

# Post transaction

Validate
https://live.blockcypher.com/btc/decodetx/

Post for testing
https://live.blockcypher.com/btc/pushtx/

Post via API
https://www.blockcypher.com/dev/bitcoin/#push-raw-transaction-endpoint

See https://en.bitcoin.it/wiki/Transaction_broadcasting

# Generate mulitsig 2-2 address

```
const bitcoin = require('bitcoinjs-lib')
const TESTNET = bitcoin.networks.testnet
const bip32 = require('bip32')

// Get WIF from Copay Wallet xPrivKey (1-st coowner)
const t =
bip32.fromBase58('tprv8ZgxMBicQKsPexegQB353wsJRdnYgj4CimrYLaSiGoEJQ39aTa464L5c
zHLFyCrfEf3Wys6ZNhy3xLEVCyL33JJ7ZRm2ym7bSHjFtCxsq9T', TESTNET);
const derivedIndex = "m/44'/1'/0'/0/0";
const wifKey = t.derivePath(derivedIndex).toWIF();
const keyPair = bitcoin.ECPair.fromWIF(wifKey, TESTNET)

// Get WIF from Copay Wallet xPrivKey (2-st coowner)
const c =
bip32.fromBase58('tprv8ZgxMBicQKsPd18wdhpYEXeMzwXPite9m3YPw84USg3Bu6K9i35x1
8KXys25N914CrBpd2ScThXtd66EMucNxZsu6dCaRVyPf9BcboZoDhP', TESTNET);
const wifKey2 = c.derivePath(derivedIndex).toWIF();
const keyPair2 = bitcoin.ECPair.fromWIF(wifKey2, TESTNET);

// Print public keys of two participants.
console.log(keyPair.publicKey.toString('hex'))
console.log(keyPair2.publicKey.toString('hex'))

// Converting two aquired public keys to a buffer
const pubkeys = [
'03541cd1f09a147f3d9b8973cc0e758445235169137af44e6cb1e891202ed3cead',
'02f9029d6ccdea678c54f7da77789fd240661ea6376f8b06e110d979061f0029a9'
].map((hex) => Buffer.from(hex, 'hex'))

// p2ms (pay-to-multisig) script 2 from 2
const p2ms = bitcoin.payments.p2ms({ m: 2, pubkeys,TESTNET });
p2ms.network = TESTNET;

// Getting p2sh multisig address
const { address } = bitcoin.payments.p2sh({
redeem: p2ms, TESTNET
})

// Print aquired address
console.log(address)
```

# Mulitsig 2-2 transaction

```
// Import key pairs for owner 1 and 2
const t =
bip32.fromBase58('tprv8ZgxMBicQKsPexegQB353wsJRdnYgj4CimrYLaSiGoEJQ39aTa4
64L5c
zHLFyCrfEf3Wys6ZNhy3xLEVCyL33JJ7ZRm2ym7bSHjFtCxsq9T', TESTNET);
const derivedIndex = "m/44'/1'/0'/0/0";
const wifKey1 = t.derivePath(derivedIndex).toWIF();
const keyPair1 = bitcoin.ECPair.fromWIF(wifKey1, TESTNET)
keyPair1.network = TESTNET
const c =
bip32.fromBase58('tprv8ZgxMBicQKsPd18wdhpYEXeMzwXPite9m3YPw84USg3Bu6K9i
35x1
8KXys25N914CrBpd2ScThXtd66EMucNxZsu6dCaRVyPf9BcboZoDhP', TESTNET);
const wifKey2 = c.derivePath(derivedIndex).toWIF();
const keyPair2 = bitcoin.ECPair.fromWIF(wifKey2, TESTNET);
keyPair2.network = TESTNET
const pubkeys = [
  '03541cd1f09a147f3d9b8973cc0e75845235169137af44e6cb1e891202ed3cead',
  '02f9029d6ccdea678c54f7da77789fd240661ea6376f8b06e110d979061f0029a9'
].map((hex) => Buffer.from(hex, 'hex'));
const p2ms = bitcoin.payments.p2ms({ m: 2, pubkeys,TESTNET });
const p2sh = bitcoin.payments.p2sh({
  redeem: p2ms, network:TESTNET
});
```

# Mulitsig 2-2 transaction

```
// Create new transaction
const txb = new bitcoin.TransactionBuilder(TESTNET)

// Hash of a transaction with transfer to multisig wallet (get from blockchain explorer)
const txId = '27a06b2c3c532fecc0452069d916fce89729ad74fbf45501a84dcbed98e17a47'

// Use output of previous transaction with index 0 as an input of new transaction
txb.addInput(txId, 0)

// Target address and sum of transfer.
txb.addOutput('mvB97Yiso2J1Gge6qvS2au8CwB2BTBQbTM', 4900000)

// Signing transaction with given pay-to-script-hash script
txb.sign(0, keyPair1, p2sh.redeem.output)
txb.sign(0, keyPair2, p2sh.redeem.output)
```

# SegWit Example (library v. 5.2)

```
const bitcoin = require('bitcoinjs-lib')
const BIP32Factory = require('bip32').default
const TESTNET = bitcoin.networks.testnet
const keyPair = bitcoin.ECPair.makeRandom(TESTNET)


const { address } = bitcoin.payments.p2pkh({
  pubkey: keyPair.publicKey,
  network: TESTNET })
console.log(address)


import('tiny-secp256k1').then(ecc => BIP32Factory(ecc)).then(bip32 => {
  let t = bip32.fromBase58('xprv9s...')

  // BIP44 path, bitcoin, account 0, external address
  // See https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki
  // m / purpose' / coin_type' / account' / change / address_index
  // set change = 1 for change address
  // address_index starts from 0 and up
  const derivedIndex = "m/44'/1'/0'/1/2"; // Change address tb1qttsmxju7qhjs33mfxmnysnhjvy5me449a4elsv
  // Override network to be test network
  t.network = TESTNET
  const wifKey = t.derivePath(derivedIndex).toWIF();
  const keyPair2 = bitcoin.ECPair.fromWIF(wifKey, TESTNET); // Sign transaction with this key.
  console.log('WIF_In=',keyPair2.toWIF())


  const p2wpkh = bitcoin.payments.p2wpkh({pubkey: keyPair2.publicKey, network: TESTNET})
  console.log('Previous output script:')
  console.log(p2wpkh.output.toString('hex'))

  // Searching for unspent address tb1qttsmxju7qhjs33mfxmnysnhjvy5me449a4elsv
  console.log('addressFrom:')
  console.log(p2wpkh.address)
```

```
  let pubKeyHash = bitcoin.crypto.hash160(Buffer.from(keyPair2.publicKey), 'hex').toString('hex');
  console.log('PubKey Hash: ' + pubKeyHash);

  const psbt = new bitcoin.Psbt({network: TESTNET})
  .addInput({
    hash: 'eab998c93bbbacc988a6b02931304305b2aae4ff753ef558043a92fa8aba7684', // TX_ID
    index: 0, // TX_VOUT unspend address tb1qttsmxju7qhjs33mfxmnysnhjvy5me449a4elsv
    witnessUtxo: {
      script: Buffer.from('0014' + pubKeyHash, 'hex'),
      value: 1276328
    },
  })
  .addOutput({
    address: address,
    value: 276328,
  })

  psbt.signInput(0, keyPair2)
  psbt.validateSignaturesOfInput(0)
  psbt.finalizeAllInputs()
  console.log('Transaction hexadecimal:')
  console.log(psbt.extractTransaction(true).toHex()) // Put true here to ignore fee warning

})
```

# Homework

1. Install NPM and NodeJS.

2. Install bitcoinjs 5.2 library, read the examples.

3. Start to do Lab 2.