Electrum Documentation

Release 2.10

Thomas Voegtlin

Apr 10, 2018

Contents

Electrum is a lightweight Bitcoin wallet.

CHAPTER 1

GUI and beginners

1.1 Frequently Asked Questions

1.1.1 How does Electrum work?

Electrum's focus is speed, with low resource usage and simplifying Bitcoin. Startup times are instant because it operates in conjunction with high-performance servers that handle the most complicated parts of the Bitcoin system.

1.1.2 Does Electrum trust servers?

Not really; the Electrum client never sends private keys to the servers. In addition, it verifies the information reported by servers, using a technique called *Simple Payment Verification*

1.1.3 What is the seed?

The seed is a random phrase that is used to generate your private keys.

Example:

```
slim sugar lizard predict state cute awkward asset inform blood civil sugar
```

Your wallet can be entirely recovered from its seed. For this, select the "restore wallet" option in the startup.

1.1.4 How secure is the seed?

The seed phrase created by Electrum has 132 bits of entropy. This means that it provides the same level of security as a Bitcoin private key (of length 256 bits). Indeed, an elliptic curve key of length n provides n/2 bits of security.

1.1.5 I have forgotten my password. What can I do?

It is not possible to recover your password. However, you can restore your wallet from its seed phrase and choose a new password. If you lose both your password and your seed, there is no way to recover your money. This is why we ask you to save your seed phrase on paper.

To restore your wallet from its seed phrase, create a new wallet, select the type, choose “I already have a seed” and proceed to input your seed phrase.

1.1.6 My transaction has been unconfirmed for a long time. What can I do?

Bitcoin transactions become “confirmed” when miners accept to write them in the Bitcoin blockchain. In general, the speed of confirmation depends on the fee you attach to your transaction; miners prioritize transactions that pay the highest fees.

Recent versions of Electrum use “dynamic fees” in order to make sure that the fee you pay with your transaction is adequate. This feature is enabled by default in recent versions of Electrum.

If you have made a transaction that is unconfirmed, you can:

- Wait for a long time. Eventually, your transaction will either be confirmed or cancelled. This might take several days.
- Increase the transaction fee. This is only possible for “replaceable” transactions. To create this type of transaction, you must have checked “Replaceable” on the send tab before sending the transaction. If you’re not seeing the “Replaceable” option on the send tab go to Tools menu > Preferences > Fees tab and set “Propose Replace-By-Fee” to “Always”. Transactions that are replaceable have the word “Replaceable” in the date column on the history tab. To increase the fee of a replaceable transaction right click on its entry on the history tab and choose “Increase Fee”. Set an appropriate fee and click on “OK”. A window will popup with the unsigned transaction. Click on “Sign” and then “Broadcast”.
- Create a “Child Pays for Parent” transaction. A CPFP is a new transaction that pays a high fee in order to compensate for the small fee of its parent transaction. It can be done by the recipient of the funds, or by the sender, if the transaction has a change output. To create a CPFP transaction right click on the unconfirmed transaction on the history tab and choose “Child pays for parent”. Set an appropriate fee and click on “OK”. A window will popup with the unsigned transaction. Click on “Sign” and then “Broadcast”.

1.1.7 What does it mean to “freeze” an address in Electrum?

When you freeze an address, the funds in that address will not be used for sending bitcoins. You cannot send bitcoins if you don’t have enough funds in the non-frozen addresses.

1.1.8 How is the wallet encrypted?

Electrum uses two separate levels of encryption:

- Your seed and private keys are encrypted using AES-256-CBC. The private keys are decrypted only briefly, when you need to sign a transaction; for this you need to enter your password. This is done in order to minimize the amount of time during which sensitive information is unencrypted in your computer’s memory.
- In addition, your wallet file may be encrypted on disk. Note that the wallet information will remain unencrypted in the memory of your computer for the duration of your session. If a wallet is encrypted, then its password will be required in order to open it. Note that the password will not be kept in memory; Electrum does not need it in order to save the wallet on disk, because it uses asymmetric encryption (ECIES).

Wallet file encryption is activated by default since version 2.8. It is intended to protect your privacy, but also to prevent you from requesting bitcoins on a wallet that you do not control.

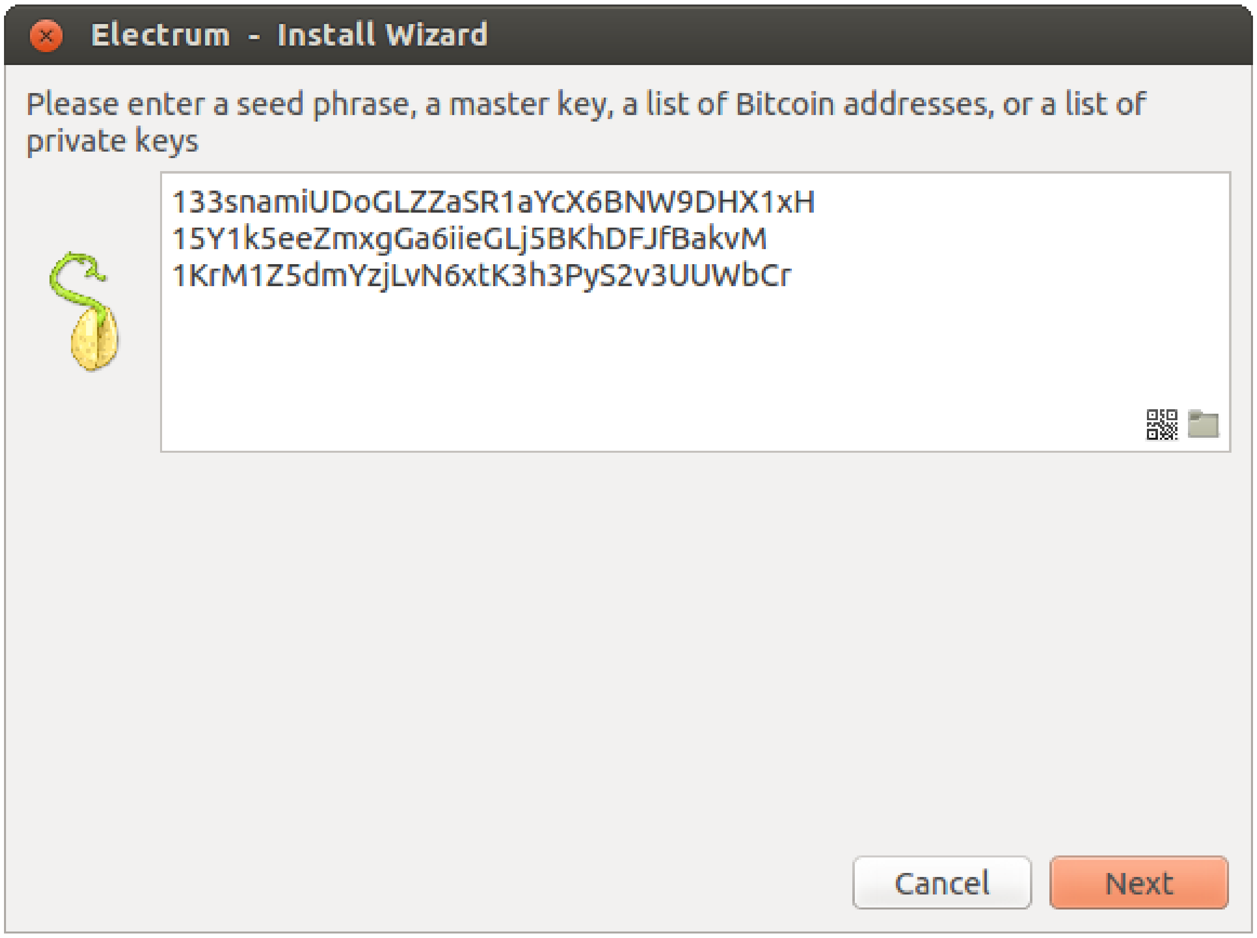
1.1.9 Does Electrum support cold wallets?

Yes, see *Cold Storage*.

1.1.10 Can I import private keys from other Bitcoin clients?

In Electrum 2.0, you cannot import private keys in a wallet that has a seed. You should sweep them instead.

If you want to import private keys and not sweep them, you need to create a special wallet that does not have a seed. For this, create a new wallet, select "restore", and instead of typing your seed, type a list of private keys, or a list of addresses if you want to create a watching-only wallet.



Please enter a seed phrase, a master key, a list of Bitcoin addresses, or a list of private keys

```
133snamiUDoGLZZaSR1aYcX6BNW9DHX1xH
15Y1k5eeZmxgGa6iieGLj5BKhDFJfBakvM
1KrM1Z5dmYzjLvN6xtK3h3PyS2v3UUWbCr
```

You will need to back up this wallet, because it cannot be recovered from a seed.

1.1.11 Can I sweep private keys from other Bitcoin clients?

Sweeping private keys means to send all the bitcoins they control to an existing address in your wallet. The private keys you sweep do not become a part of your wallet. Instead, all the bitcoins they control are sent to an address that has been deterministically generated from your wallet seed.

To sweep private keys, go to the Wallet menu -> Private Keys -> Sweep. Enter the private keys in the appropriate field. Leave the "Address" field unchanged. That is the destination address and it will be from your existing electrum wallet. Click on "Sweep". It'll now take you to the send tab where you can set an appropriate fee and then click on "Send" to send the coins to your wallet.

1.1.12 Where is my wallet file located?

The default wallet file is called default_wallet, which is created when you first run the application and is located in the /wallets folder.

On Windows:

- Show hidden files
- Go to \Users\YourUserName\AppData\Roaming\Electrum\wallets (or %APPDATA%\Electrum\wallets)

On Mac:

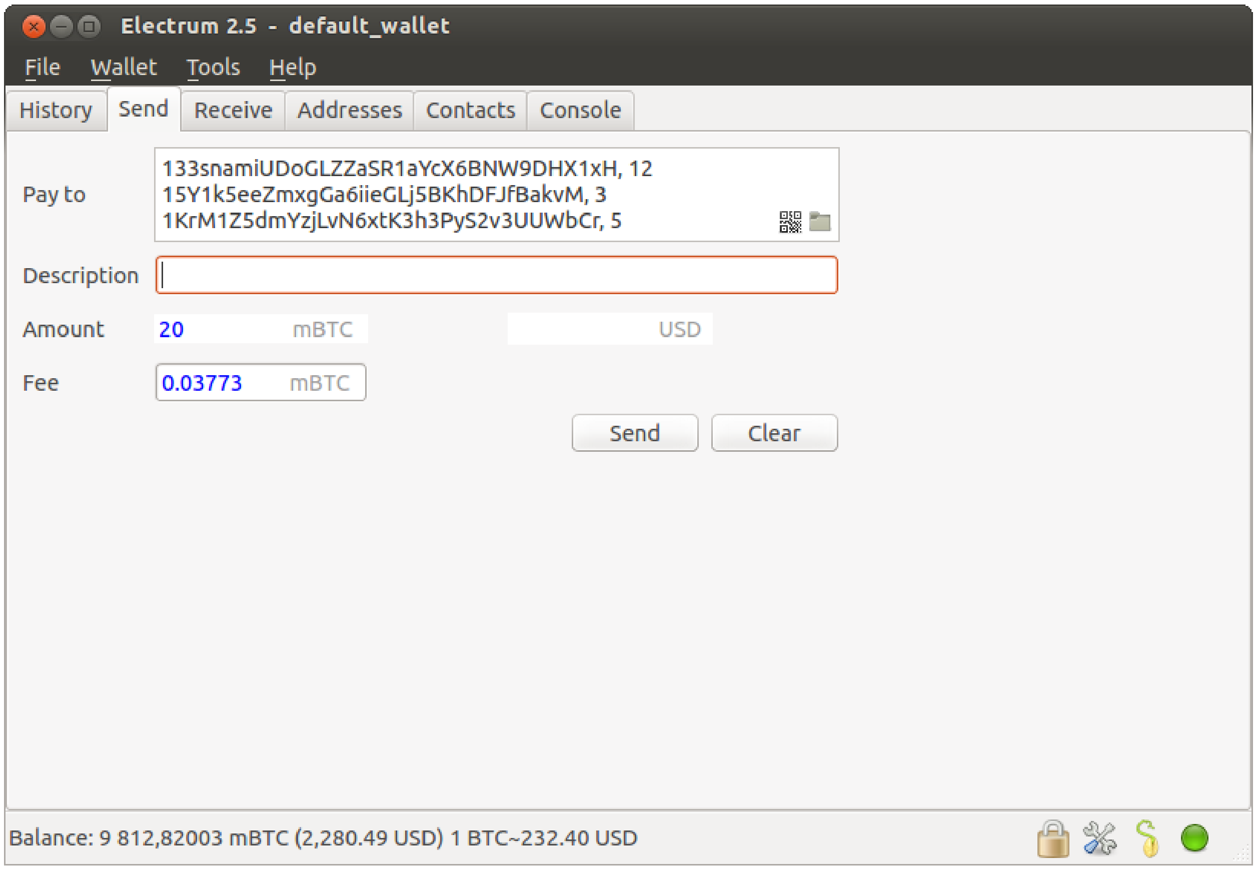
- Open Finder
- Go to folder (shift+cmd+G) and type ~/.electrum

On Linux:

- Home Folder
- Go -> Location and type ~/.electrum

1.1.13 Can I do bulk payments with Electrum?

You can create a transaction with several outputs. In the GUI, type each address and amount on a line, separated by a comma.


Electrum 2.5 - default_wallet
File Wallet Tools Help
History Send Receive Addresses Contacts Console
Send Clear

Pay to
133snamiUDoGLZZaSR1aYcX6BNW9DHX1xH, 12
15Y1k5eeZmxgGa6iieGLj5BKhDFJfBakvM, 3
1KrM1Z5dmYzjLvN6xtK3h3PyS2v3UUWbCr, 5

Description

Amount 20 mBTC USD

Fee 0.03773 mBTC

Balance: 9 812,82003 mBTC (2,280.49 USD) 1 BTC~232.40 USD

Amounts are in the current unit set in the client. The total is shown in the GUI.

You can also import a CSV file in the "Pay to" field, by clicking on the folder icon.

1.1.14 Can Electrum create and sign raw transactions?

Electrum lets you create and sign raw transactions right from the user interface using a form.

1.1.15 Electrum freezes when I try to send bitcoins.

This might happen if you are trying to spend a large number of transaction outputs (for example, if you have collected hundreds of donations from a Bitcoin faucet). When you send Bitcoins, Electrum looks for unspent coins that are in your wallet in order to create a new transaction. Unspent coins can have different values, much like physical coins and bills.

If this happens, you should consolidate your transaction inputs by sending smaller amounts of bitcoins to one of your wallet addresses; this would be the equivalent of exchanging a stack of nickels for a dollar bill.

1.1.16 What is the gap limit?

The gap limit is the maximum number of consecutive unused addresses in your deterministic sequence of addresses. Electrum uses it in order to stop looking for addresses. In Electrum 2.0, it is set to 20 by default, so the client will get all addresses until 20 unused addresses are found.

1.1.17 How can I pre-generate new addresses?

Electrum will generate new addresses as you use them, until it hits the *gap limit*.

If you need to pre-generate more addresses, you can do so by typing wallet.create_new_address(False) in the console. This command will generate one new address. Note that the address will be shown with a red background in the address tab to indicate that it is beyond the gap limit. The red color will remain until the gap is filled.

WARNING: Addresses beyond the gap limit will not automatically be recovered from the seed. To recover them will require either increasing the client's gap limit or generating new addresses until the used addresses are found.

If you wish to generate more than one address, you can use a "for" loop. For example, if you wanted to generate 50 addresses, you could do this:

```
for x in range(0, 50):
    print wallet.create_new_address(False)
```

1.1.18 How do I upgrade Electrum?

Warning: always save your wallet seed on paper before doing an upgrade.

To upgrade Electrum, just install the most recent version. The way to do this will depend on your OS.

Note that your wallet files are stored separately from the software, so you can safely remove the old version of the software if your OS does not do it for you.

Some Electrum upgrades will modify the format of your wallet files.

For this reason, it is not recommended to downgrade Electrum to an older version once you have opened your wallet file with the new version. The older version will not always be able to read the new wallet file.

The following issues should be considered when upgrading Electrum 1.x wallets to Electrum 2.x:

- Electrum 2.x will need to regenerate all of your addresses during the upgrade process. Please allow it time to complete, and expect it to take a little longer than usual for Electrum to be ready.
- The contents of your wallet file will be replaced with an Electrum 2 wallet. This means Electrum 1.x will no longer be able to use your wallet once the upgrade is complete.
- The "Addresses" tab will not show any addresses the first time you launch Electrum 2. This is expected behavior. Restart Electrum 2 after the upgrade is complete and your addresses will be available.
- Offline copies of Electrum will not show the addresses at all because it cannot synchronize with the network. You can force an offline generation of a few addresses by typing the following into the Console: wallet.synchronize(). When it's complete, restart Electrum and your addresses will once again be available.

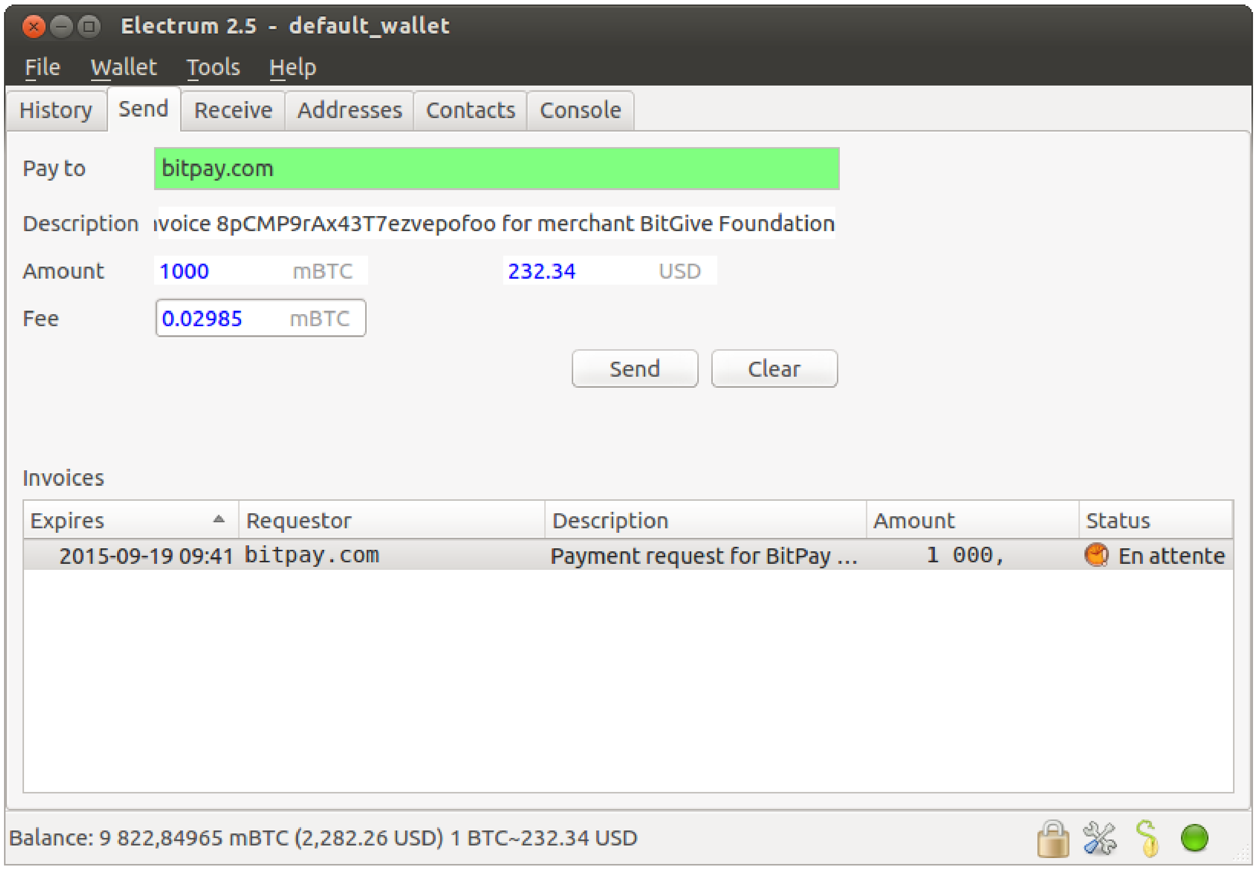
1.2 Invoices

Invoices are Payment Requests signed by their requestor.

When you click on a bitcoin: link, a URL is passed to electrum:

```
electrum "bitcoin:1KLxqw4MA5NkG6YP1N4S14akDFCP1vQrKu?amount=1.0&amp;r=https%3A%2F
↪%2Fbitpay.com%2Fi%2FXxaGtEpRSqckRnhsjZwtrA"
```

This opens the send tab with the payment request:


Electrum 2.5 - default_wallet
File Wallet Tools Help
History Send Receive Addresses Contacts Console
Send Clear

Pay to: bitpay.com

Description: ıvoice 8pCMP9rAx43T7ezvepofoo for merchant BitGive Foundation

Amount: 1000 mBTC 232.34 USD

Fee: 0.02985 mBTC

Invoices

Expires	Requestor	Description	Amount	Status
2015-09-19 09:41	bitpay.com	Payment request for BitPay ...	1 000,	En attente

Balance: 9 822,84965 mBTC (2,282.26 USD) 1 BTC~232.34 USD

The green color in the “Pay To” field means that the payment request was signed by bitpay.com’s certificate, and that Electrum verified the chain of signatures.

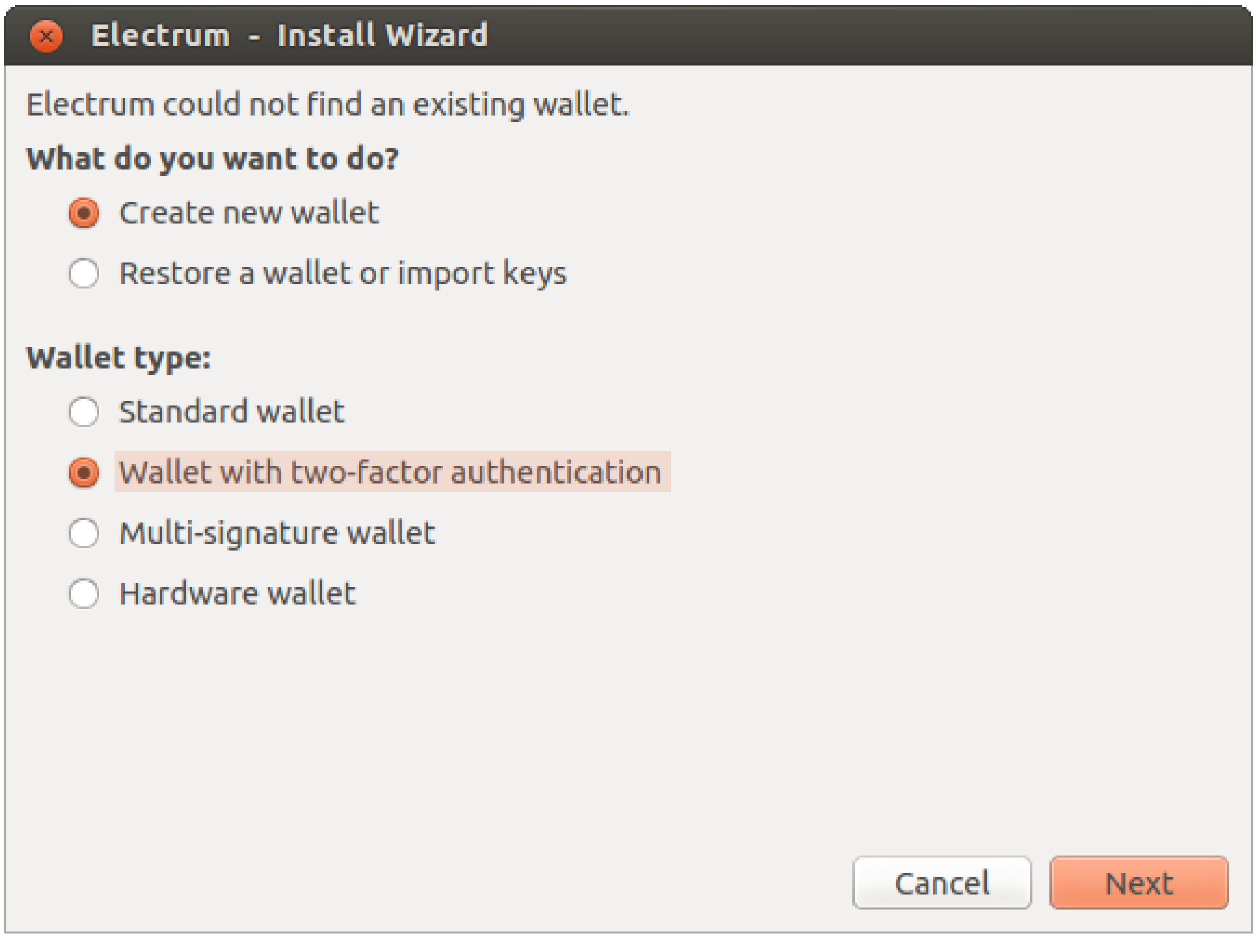
Note that the “send” tab contains a list of invoices and their status.

1.3 Two Factor Authentication

Electrum offers two-factor authenticated wallets, with a remote server acting to co-sign transactions, adding another level of security in the event of your computer being compromised.

The remote server in question is a service offered by TrustedCoin. Here is a guide on how it works.

1.3.1 Creating a Wallet



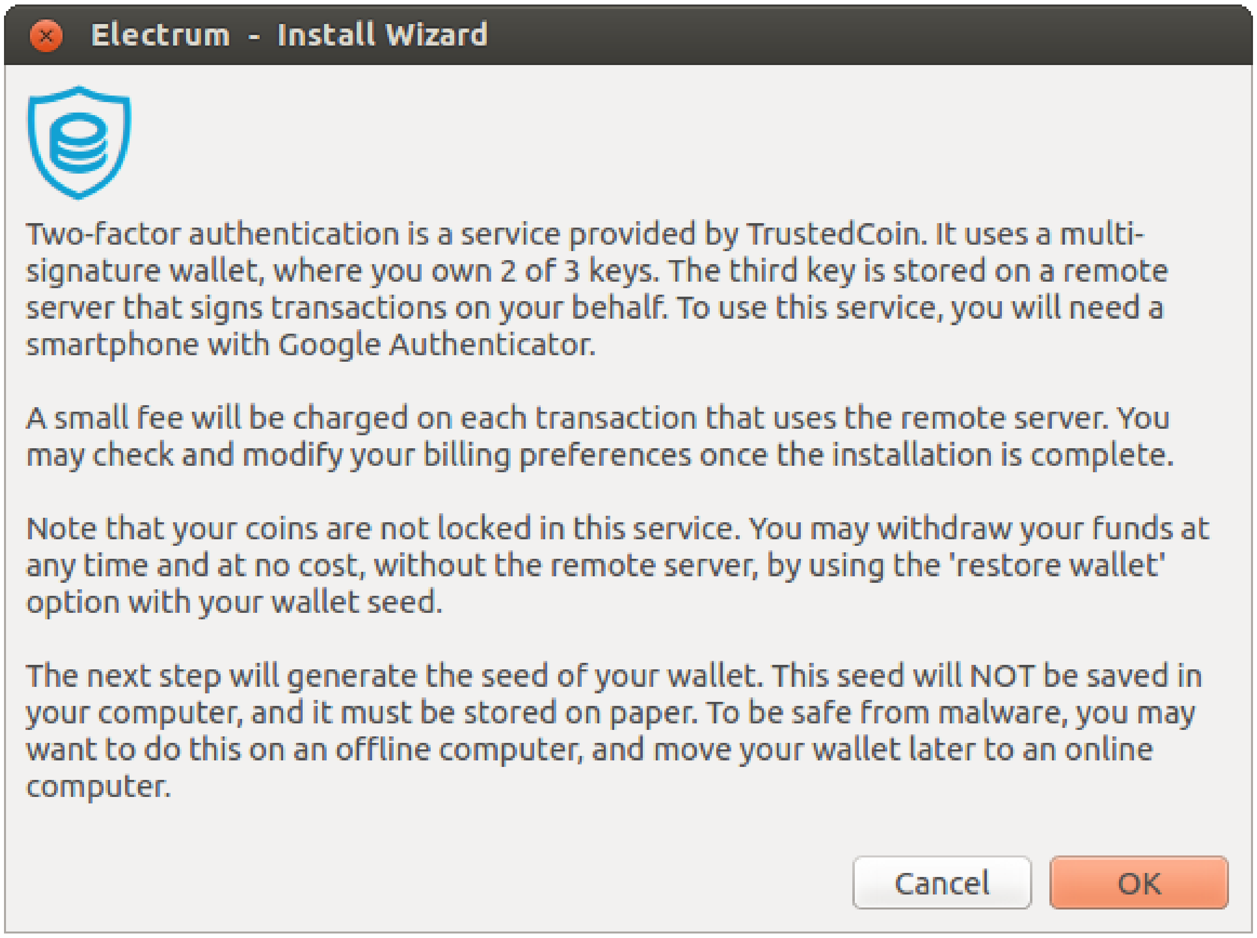
Electrum could not find an existing wallet.

What do you want to do?

- [✓ Create new wallet]
- Restore a wallet or import keys

Wallet type:

- Standard wallet
- [✓ Wallet with two-factor authentication]
- Multi-signature wallet
- Hardware wallet



Two-factor authentication is a service provided by TrustedCoin. It uses a multi-signature wallet, where you own 2 of 3 keys. The third key is stored on a remote server that signs transactions on your behalf. To use this service, you will need a smartphone with Google Authenticator.

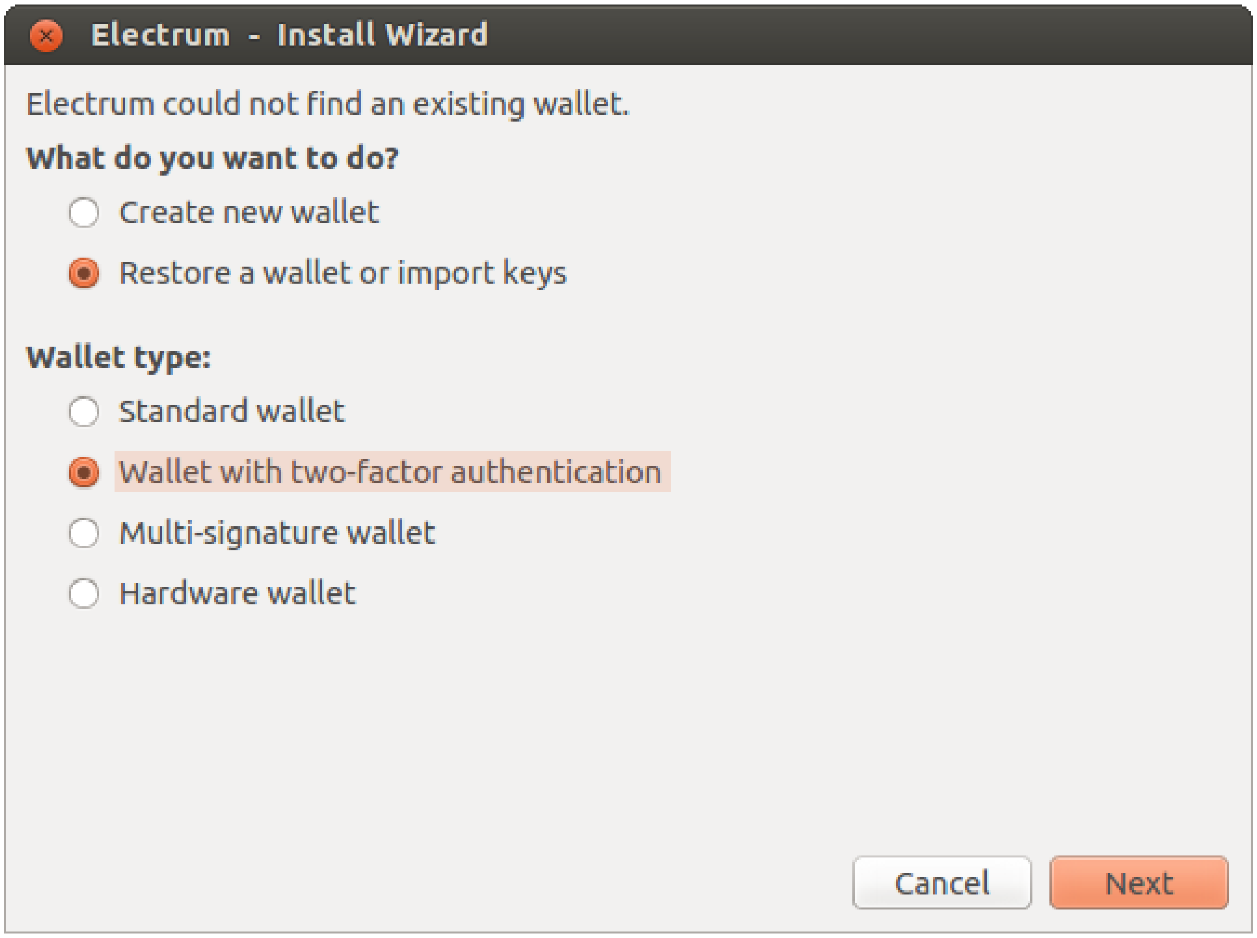
A small fee will be charged on each transaction that uses the remote server. You may check and modify your billing preferences once the installation is complete.

Note that your coins are not locked in this service. You may withdraw your funds at any time and at no cost, without the remote server, by using the 'restore wallet' option with your wallet seed.

The next step will generate the seed of your wallet. This seed will NOT be saved in your computer, and it must be stored on paper. To be safe from malware, you may want to do this on an offline computer, and move your wallet later to an online computer.

1.3.2 Restoring from seed

Even if TrustedCoin is compromised or taken offline, your coins are secure as long as you still have the seed of your wallet. Your seed contains two master private keys in a 2-of-3 security scheme. In addition, the third master public key can be derived from your seed, ensuring that your wallet addresses can be restored. In order to restore your wallet from seed, select “wallet with two factor authentication”, as this tells Electrum to use this special variety of seed for restoring your wallet.



Electrum - Install Wizard

Electrum could not find an existing wallet.

What do you want to do?

- ○ Create new wallet
- ● Restore a wallet or import keys

Wallet type:

- ○ Standard wallet
- ● Wallet with two-factor authentication
- ○ Multi-signature wallet
- ○ Hardware wallet

Note: The "restore" option should be used only if you no longer want to use TrustedCoin, or if there is a problem with the service. Once you have restored your wallet in this way, two of three factors are on your machine, negating the special protection of this type of wallet.

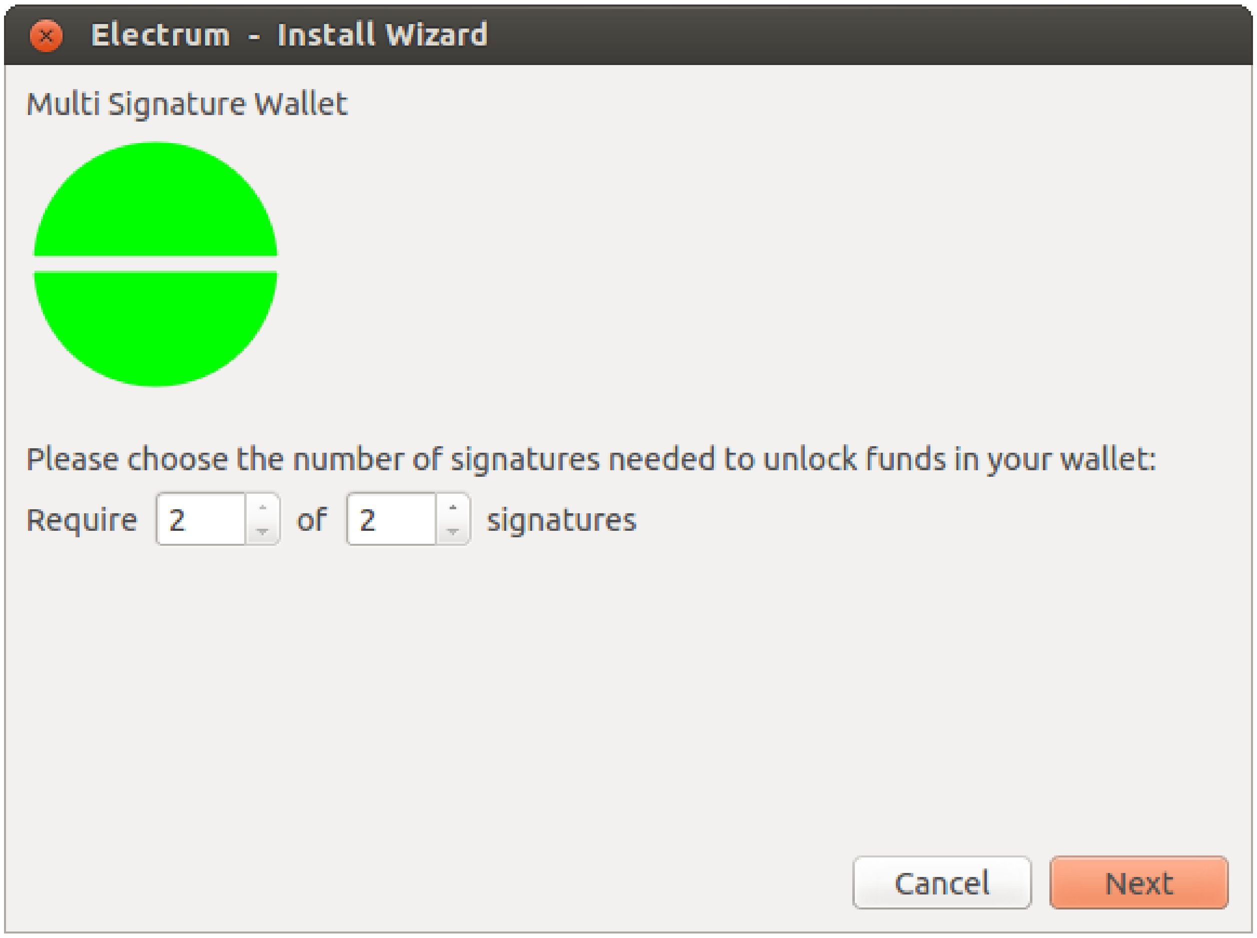
1.4 Multisig Wallets

This tutorial shows how to create a 2 of 2 multisig wallet. A 2 of 2 multisig consists of 2 separate wallets (usually on separate machines and potentially controlled by separate people) that have to be used in conjunction in order to access the funds. Both wallets have the same set of Addresses.

- A common use-case for this is if you want to collaboratively control funds: maybe you and your friend run a company together and certain funds should only be spendable if you both agree.
- Another one is security: One of the wallets can be on your main machine, while the other one is on a offline machine. That way you make it very hard for an attacker or malware to steal your coins.

1.4.1 Create a pair of 2-of-2 wallets

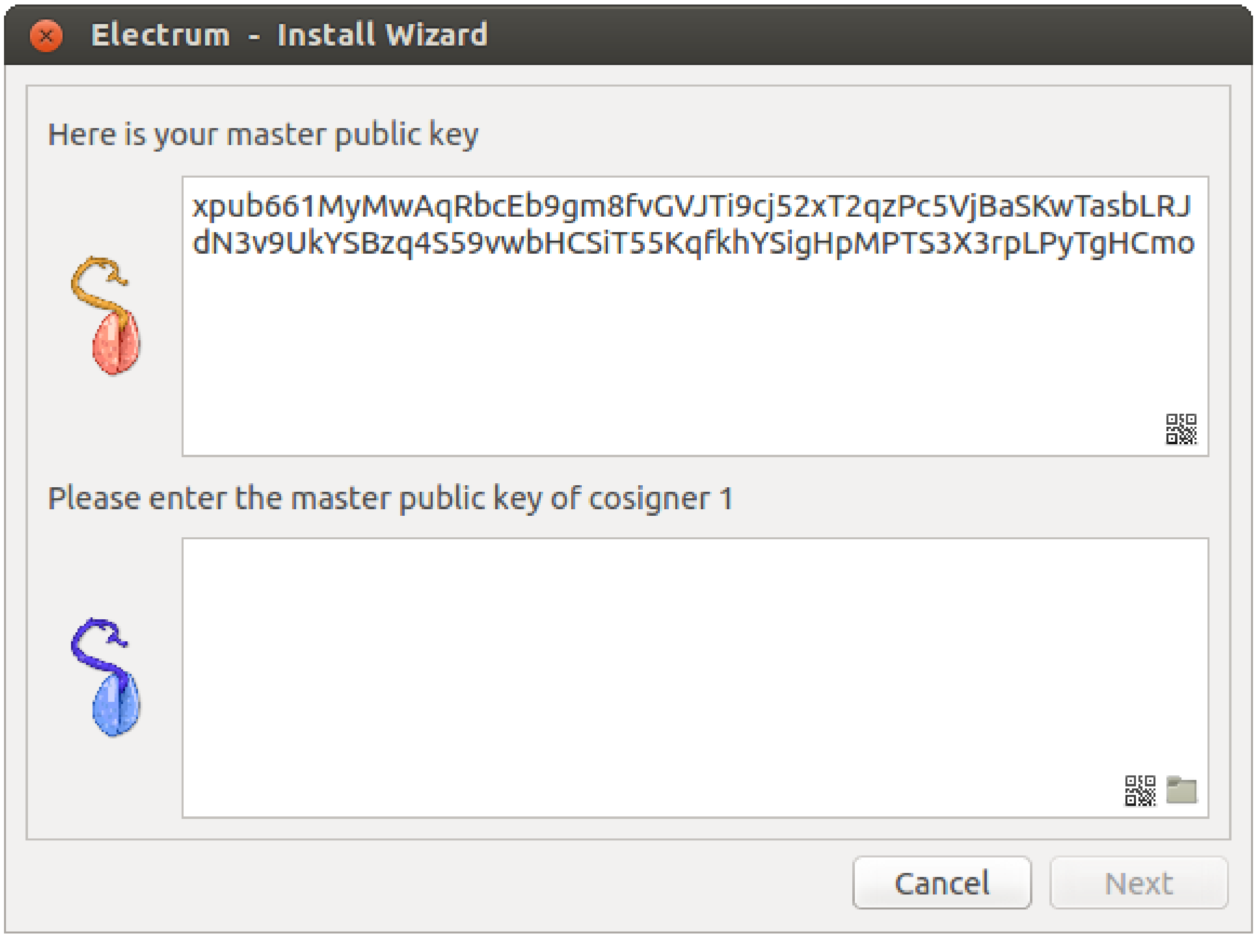
Each cosigner needs to do this: In the menu select File->New, then select "Multi-signature wallet". On the next screen, select 2 of 2.



Please choose the number of signatures needed to unlock funds in your wallet:

Require 2 of 2 signatures

After generating a seed (keep it safely!) you will need to provide the master public key of the other wallet.



Here is your master public key

xpub661MyMwAqRbcEb9gm8fvGVJTi9cj52xT2qzPc5VjBaSKwTasbLRJdN3v9UkYSBzq4S59vwbHCSiT55KqfkhYSigHpMPTS3X3rpLPyTgHCmo

Please enter the master public key of cosigner 1

Put the master public key of the other wallet into the lower box. Of course when you create the other wallet, you put the master public key of this one.

You will need to do this in parallel for the two wallets. Note that you can press cancel during this step, and reopen the file later.

1.4.2 Receiving

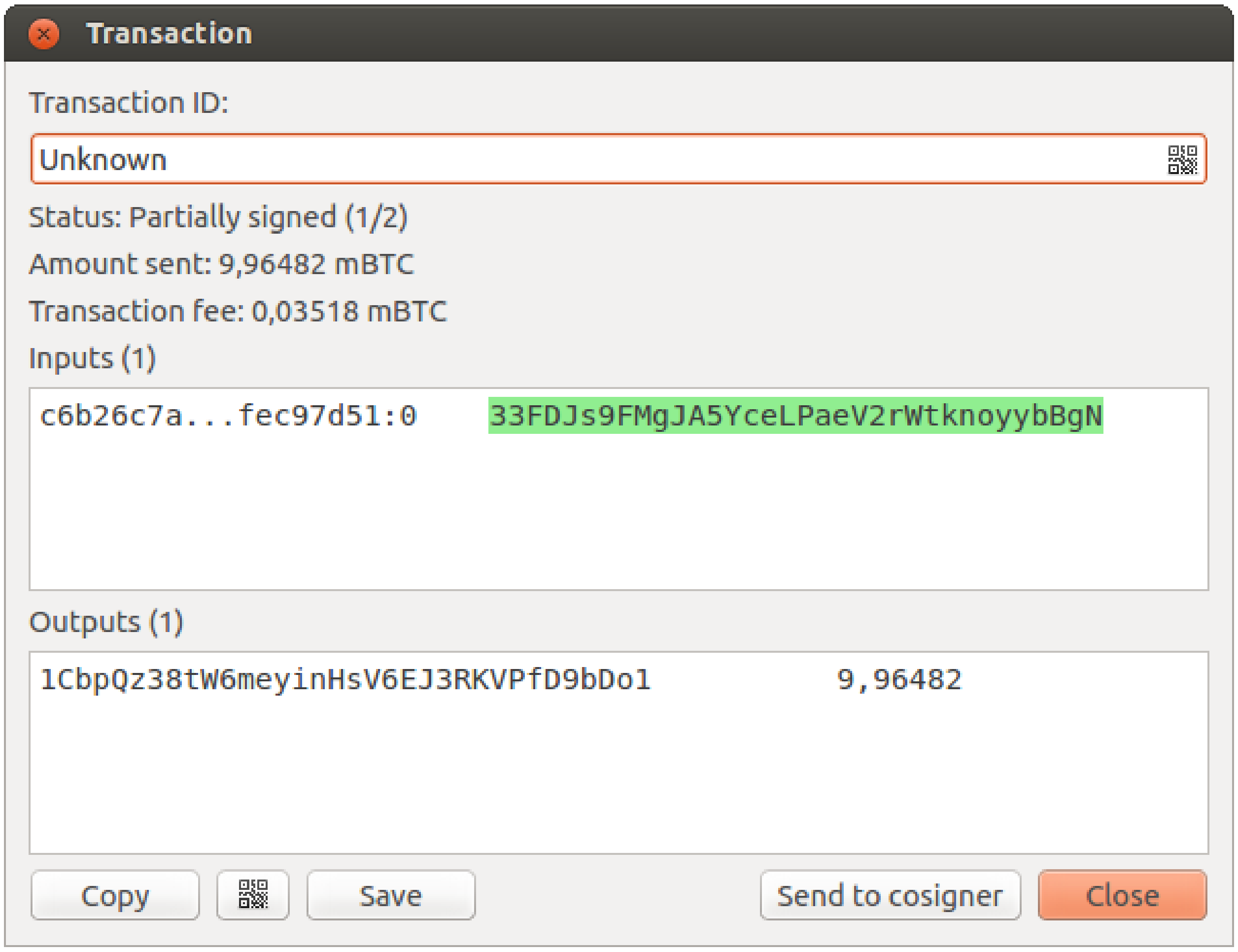
Check that both wallets generate the same set of Addresses. You can now send to these Addresses (note they start with a "3") with any wallet that can send to P2SH Addresses.

1.4.3 Spending

To spend coins from a 2-of-2 wallet, two cosigners need to sign a transaction collaboratively.

To accomplish this, create a transaction using one of the wallets (by filling out the form on the "send" tab)

After signing, a window is shown with the transaction details.



Transaction ID:

Unknown

Status: Partially signed (1/2)

Amount sent: 9,96482 mBTC

Transaction fee: 0,03518 mBTC

Inputs (1)

```
c6b26c7a...fec97d51:0     33FDJs9FMgJA5YceLPaeV2rWtknoyybBgN
```

Outputs (1)

```
1CbpQz38tW6meyinHsV6EJ3RKVPfD9bDo1          9,96482
```

The transaction has to be sent to the second wallet.

For this you have multiple options:

- you can transfer the file on a usb stick
- you can use QR codes
- you can use a remote server, with the CosignerPool plugin.

Transfer a file

You can save the partially signed transaction to a file (using the "save" button), transfer that to the machine where the second wallet is running (via usb stick, for example) and load it there (using Tools -> Load transaction -> from file)

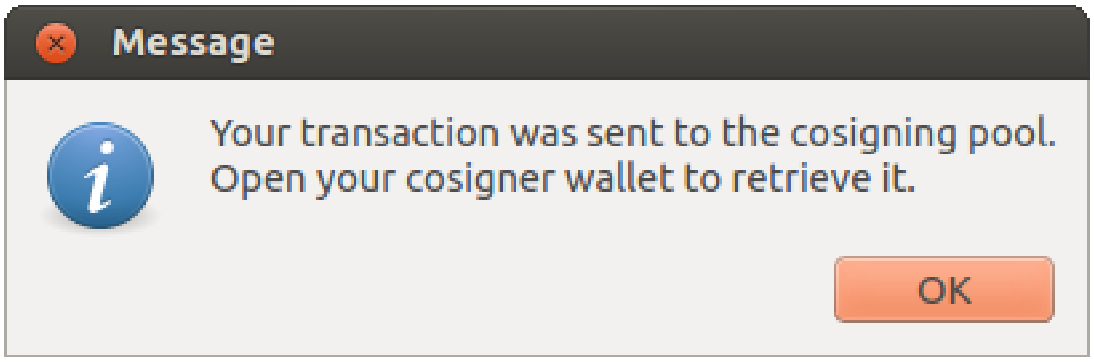
Use QR-Code

There's also a button showing a qr-code icon. Clicking that will display a qr-code containing the transaction that can be scanned into the second wallet (Tools -> Load Transaction -> From QR Code)

Use the Cosigner Pool Plugin

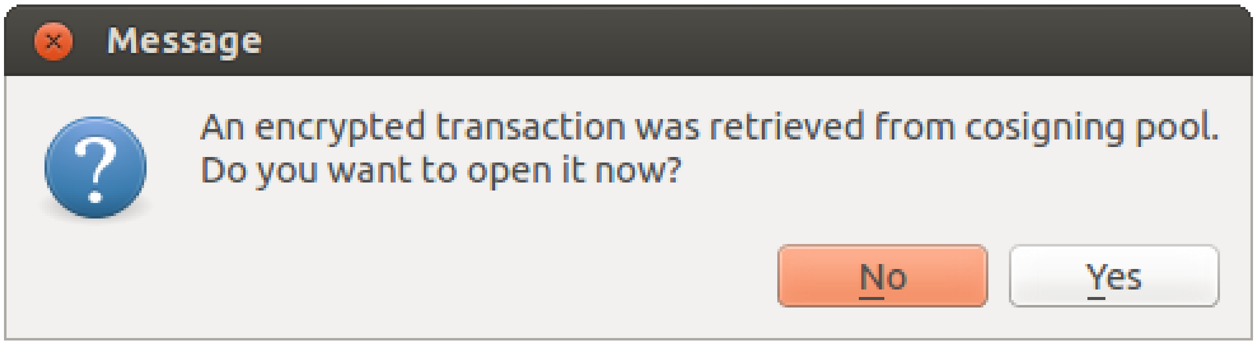
For this to work the Plugin "Cosigner Pool" needs to be enabled (Tools -> Plugins) with both wallets.

Once the plugin is enabled, you will see a button labeled "Send to cosigner". Clicking it sends the partially signed transaction to a central server. Note that the transaction is encrypted with your cosigner's master public key.



Your transaction was sent to the cosigning pool.
Open your cosigner wallet to retrieve it.

When the cosigner wallet is started, it will get a notification that a partially signed transaction is available:



An encrypted transaction was retrieved from cosigning pool.
Do you want to open it now?

The transaction is encrypted with the cosigner's master public key; the password is needed to decrypt it.

With all of the above methods, you can now add the seconds signature the the transaction (using the "sign" button). It will then be broadcast to the network.

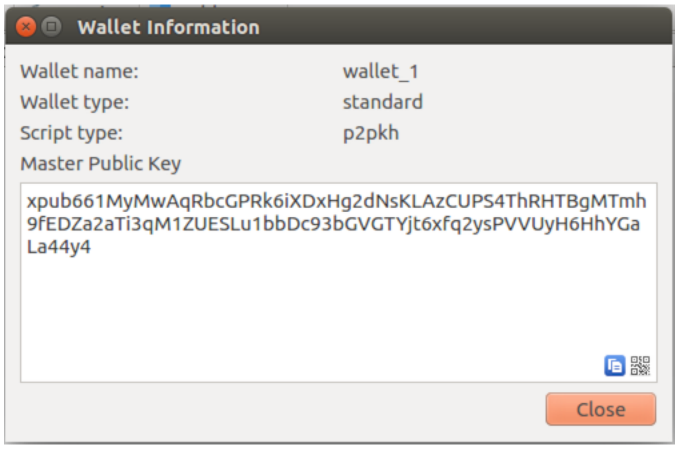
1.5 Cold Storage

This document shows how to create an offline wallet that holds your Bitcoins and a watching-only online wallet that is used to view its history and to create transactions that have to be signed with the offline wallet before being broadcast on the online one.

1.5.1 Create an offline wallet

Create a wallet on an offline machine, as per the usual process (file -> new) etc.

After creating the wallet, go to Wallet -> Information.



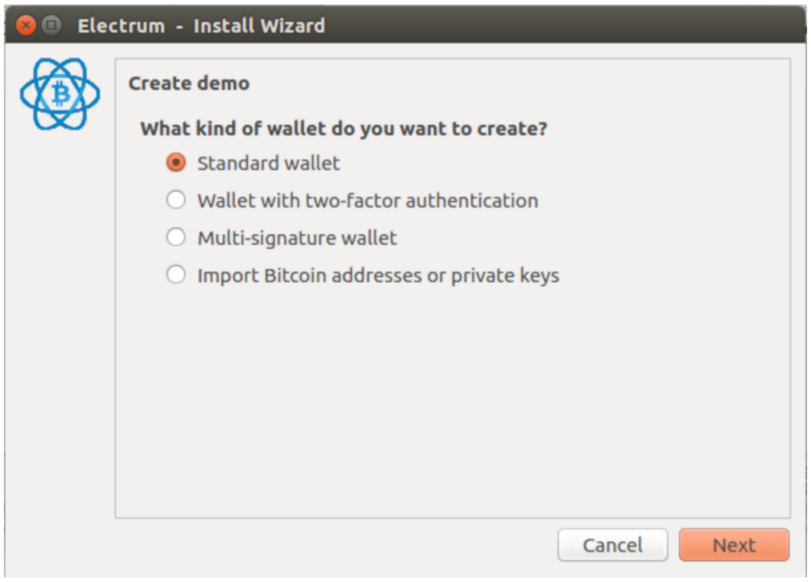
Wallet name: wallet_1
Wallet type: standard
Script type: p2pkh
Master Public Key

```
xpub661MyMwAqRbcGPRk6iXDxHg2dNsKLAzCUPS4ThRHTBgMTmh
9fEDZa2aTi3qM1ZUESLu1bbDc93bGVGTYjt6xfq2ysPVVUyH6HhYGa
La44y4
```

The Master Public Key of your wallet is the string shown in this popup window. Transfer that key to your online machine somehow.

1.5.2 Create a watching-only version of your wallet

On your online machine, open up Electrum and select File -> New/Restore. Enter a name for the wallet and select "Standard wallet".

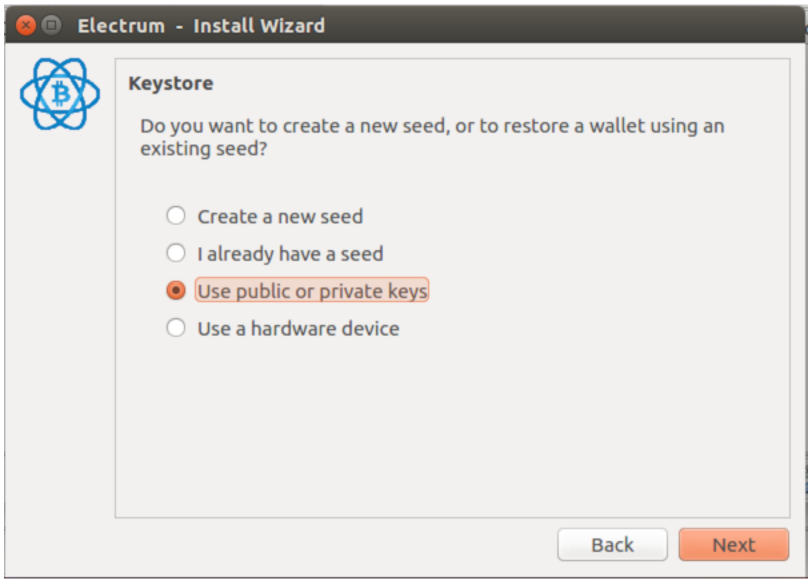


Create demo

What kind of wallet do you want to create?

[✓ Standard wallet]
Wallet with two-factor authentication
Multi-signature wallet
Import Bitcoin addresses or private keys

Select "Use public or private keys"

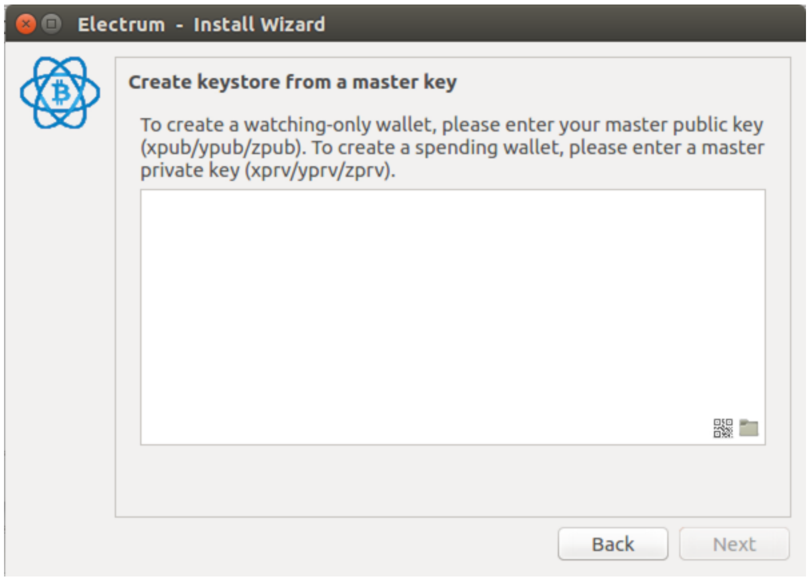


Keystore

Do you want to create a new seed, or to restore a wallet using an existing seed?

- ( ) Create a new seed
- ( ) I already have a seed
- [✓ Use public or private keys]
- ( ) Use a hardware device

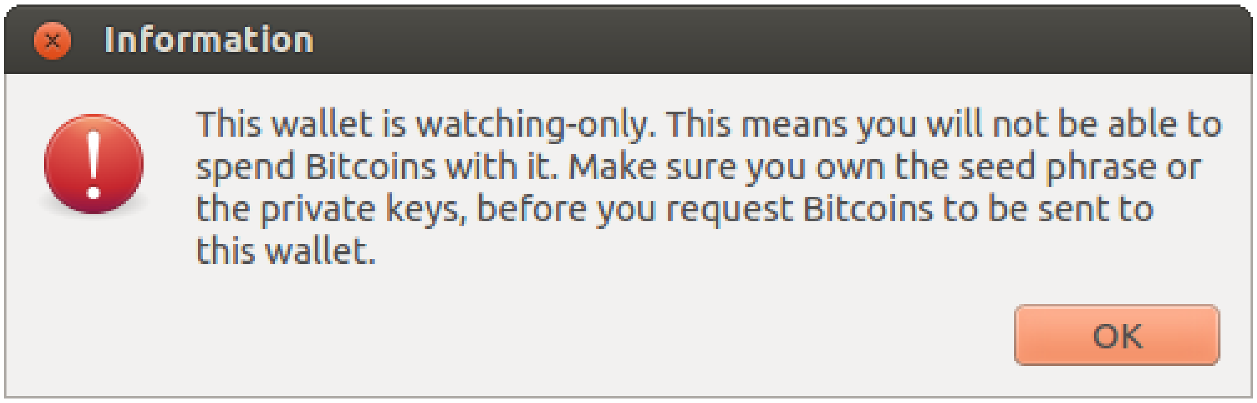
Paste your master public key in the box.



Create keystore from a master key

To create a watching-only wallet, please enter your master public key (xpub/ypub/zpub). To create a spending wallet, please enter a master private key (xprv/yprv/zprv).

Click Next to complete the creation of your wallet. When you're done, you should see a popup informing you that you are opening a watching-only wallet.

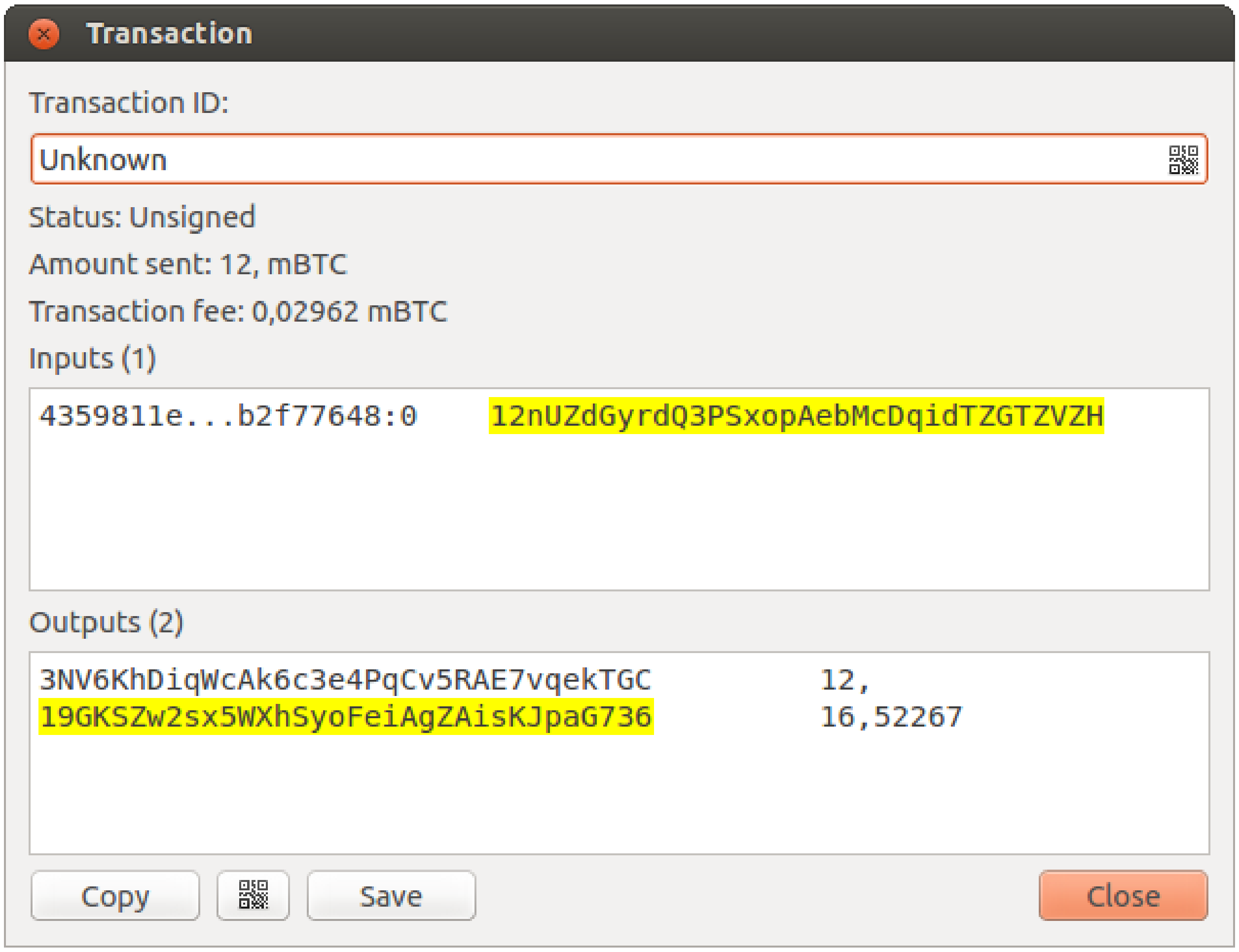


This wallet is watching-only. This means you will not be able to spend Bitcoins with it. Make sure you own the seed phrase or the private keys, before you request Bitcoins to be sent to this wallet.

Then you should see the transaction history of your cold wallet.

1.5.3 Create an unsigned transaction

Go to the "send" tab on your online watching-only wallet, input the transaction data and press "Preview". A window pops up:



Transaction ID:

Unknown

Status: Unsigned

Amount sent: 12, mBTC

Transaction fee: 0,02962 mBTC

Inputs (1)

```
4359811e...b2f77648:0     12nUZdGyrdQ3PSxopAebMcDqidTZGTZVZH
```

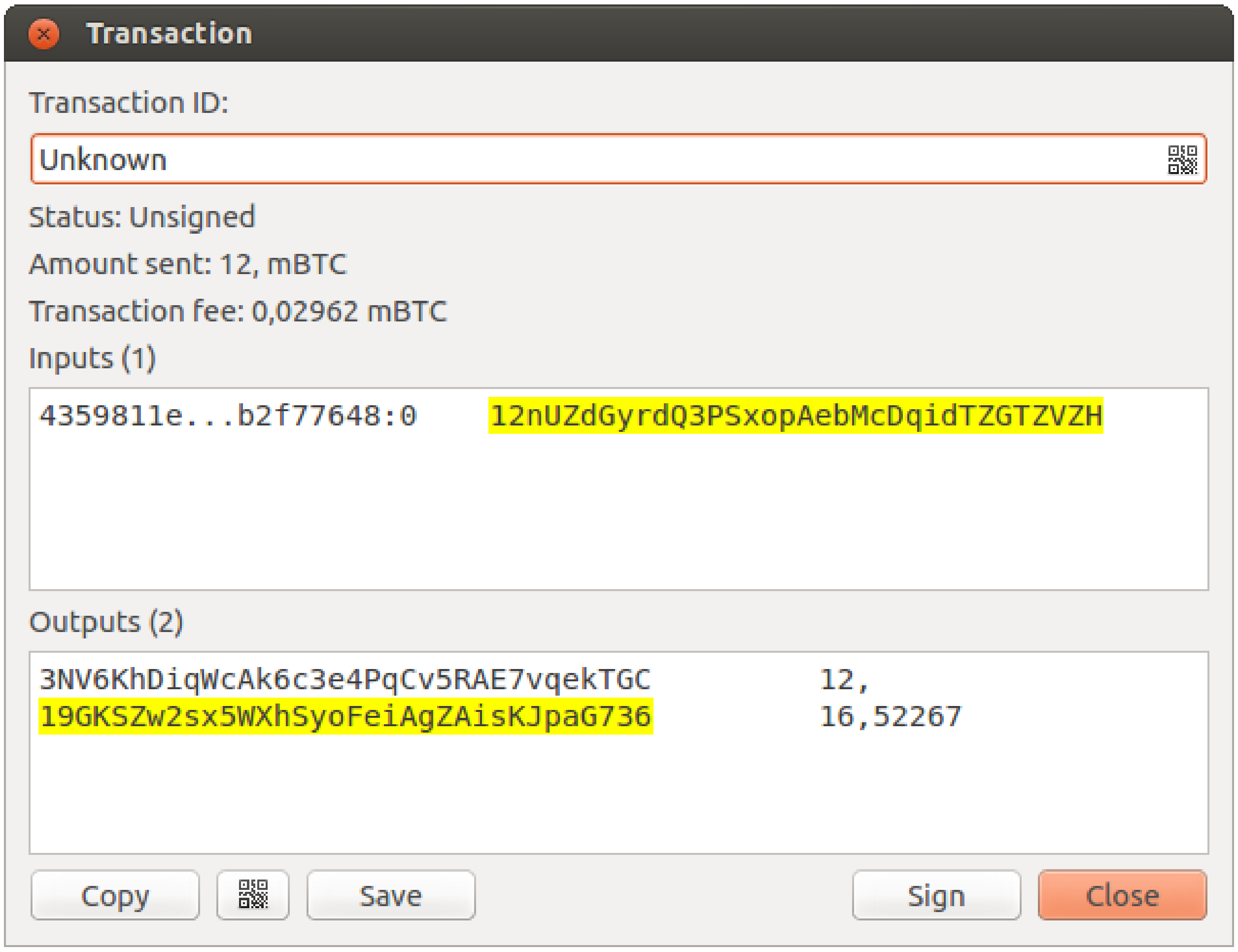
Outputs (2)

```
3NV6KhDiqWcAk6c3e4PqCv5RAE7vqekTGC          12,
19GKSZw2sx5WXhSyoFeiAgZAisKJpaG736          16,52267
```

Press "save" and save the transaction file somewhere on your computer. Close the window and transfer the transaction file to your offline machine (e.g. with a usb stick).

1.5.4 Get your transaction signed

On your offline wallet, select Tools -> Load transaction -> From file in the menu and select the transaction file created in the previous step.



Transaction ID:

Unknown

Status: Unsigned

Amount sent: 12, mBTC

Transaction fee: 0,02962 mBTC

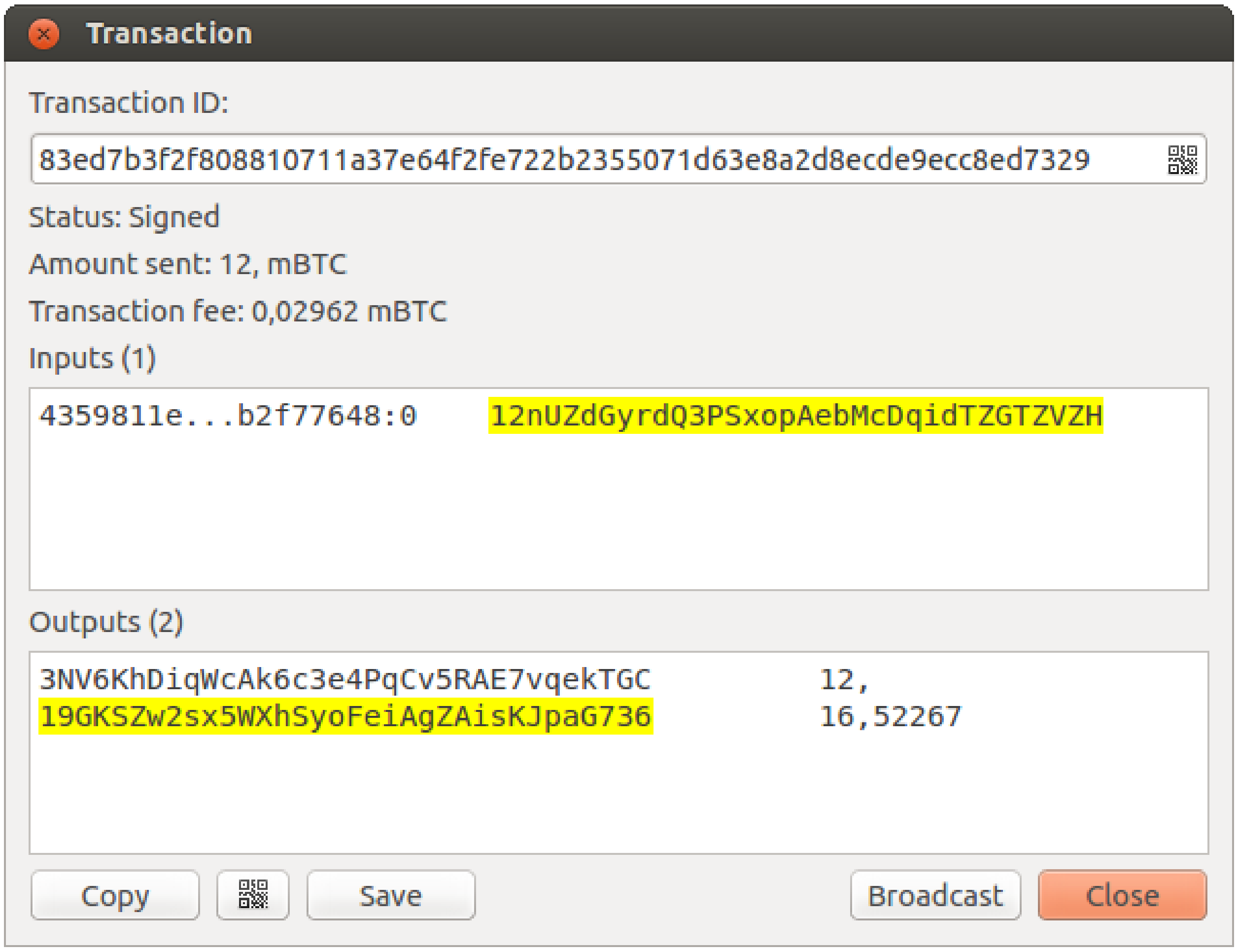
Inputs (1)

```
4359811e...b2f77648:0     12nUZdGyrdQ3PSxopAebMcDqidTZGTZVZH
```

Outputs (2)

```
3NV6KhDiqWcAk6c3e4PqCv5RAE7vqekTGC          12,
19GKSZw2sx5WXhSyoFeiAgZAisKJpaG736          16,52267
```

Press "sign". Once the transaction is signed, the Transaction ID appears in its designated field.



Transaction ID:

83ed7b3f2f808810711a37e64f2fe722b2355071d63e8a2d8ecde9ecc8ed7329

Status: Signed

Amount sent: 12, mBTC

Transaction fee: 0,02962 mBTC

Inputs (1)

```
4359811e...b2f77648:0     12nUZdGyrdQ3PSxopAebMcDqidTZGTZVZH
```

Outputs (2)

```
3NV6KhDiqWcAk6c3e4PqCv5RAE7vqekTGC           12,
19GKSZw2sx5WXhSyoFeiAgZAisKJpaG736           16,52267
```

Press save, store the file somewhere on your computer, and transfer it back to your online machine.

1.5.5 Broadcast your transaction

On your online machine, select Tools -> Load transaction -> From File from the menu. Select the signed transaction file. In the window that opens up, press “broadcast”. The transaction will be broadcasted over the Bitcoin network.

CHAPTER 2

Advanced users

2.1 Command Line

Electrum has a powerful command line. This page will show you a few basic principles.

2.1.1 Using the inline help

To see the list of Electrum commands, type:

```
electrum help
```

To see the documentation for a command, type:

```
electrum help <command>
```

2.1.2 Magic words

The arguments passed to commands may be one of the following magic words: ! ? : and -.

- The exclamation mark ! is a shortcut that means 'the maximum amount available'.

  Example:

  ```
  electrum payto 1JuiT4dM65d8vBt8qUYamnDmAMJ4MjjxRE !
  ```

  Note that the transaction fee will be computed and deducted from the amount.

- A question mark ? means that you want the parameter to be prompted.

  Example:

  ```
  electrum signmessage 1JuiT4dM65d8vBt8qUYamnDmAMJ4MjjxRE ?
  ```

- Use a colon : if you want the prompted parameter to be hidden (not echoed in your terminal).

  ```
  electrum importprivkey :
  ```

  Note that you will be prompted twice in this example, first for the private key, then for your wallet password.

- A parameter replaced by a dash - will be read from standard input (in a pipe)

  ```
  cat LICENCE | electrum signmessage 1JuiT4dM65d8vBt8qUYamnDmAMJ4MjjxRE -
  ```

2.1.3 Aliases

You can use DNS aliases in place of bitcoin addresses, in most commands.

```
electrum payto ecdsa.net !
```

2.1.4 Formatting outputs using jq

Command outputs are either simple strings or json structured data. A very useful utility is the 'jq' program. Install it with:

```
sudo apt-get install jq
```

The following examples use it.

2.1.5 Examples

Sign and verify message

We may use a variable to store the signature, and verify it:

```
sig=$(cat LICENCE| electrum signmessage 1JuiT4dM65d8vBt8qUYamnDmAMJ4MjjxRE -)
```

And:

```
cat LICENCE | electrum verifymessage 1JuiT4dM65d8vBt8qUYamnDmAMJ4MjjxRE $sig -
```

Show the values of your unspents

The 'listunspent' command returns a list of dict objects, with various fields. Suppose we want to extract the 'value' field of each record. This can be achieved with the jq command:

```
electrum listunspent | jq 'map(.value)'
```

Select only incoming transactions from history

Incoming transactions have a positive 'value' field

```
electrum history | jq '.[] | select(.value>0)'
```

Filter transactions by date

The following command selects transactions that were timestamped after a given date:

```
after=$(date -d '07/01/2015' +"%s")

electrum history | jq --arg after $after '.[] | select(.timestamp>($after|tonumber))'
```

Similarly, we may export transactions for a given time period:

```
before=$(date -d '08/01/2015' +"%s")

after=$(date -d '07/01/2015' +"%s")

electrum history | jq --arg before $before --arg after $after '.[] | select(.
↪timestamp&gt;($after|tonumber) and .timestamp&lt;($before|tonumber))'
```

Encrypt and decrypt messages

First we need the public key of a wallet address:

```
pk=$(electrum getpubkeys 1JuiT4dM65d8vBt8qUYamnDmAMJ4MjjxRE| jq -r '.[0]')
```

Encrypt:

```
cat | electrum encrypt $pk -
```

Decrypt:

```
electrum decrypt $pk ?
```

Note: this command will prompt for the encrypted message, then for the wallet password

Export private keys and sweep coins

The following command will export the private keys of all wallet addresses that hold some bitcoins:

```
electrum listaddresses --funded | electrum getprivatekeys -
```

This will return a list of lists of private keys. In most cases, you want to get a simple list. This can be done by adding a jq filer, as follows:

```
electrum listaddresses --funded | electrum getprivatekeys - | jq 'map(.[0])'
```

Finally, let us use this list of private keys as input to the sweep command:

```
electrum listaddresses --funded | electrum getprivatekeys - | jq 'map(.[0])' |␣
↪electrum sweep - [destination address]
```

2.2 Using cold storage with the command line

This page will show you how to sign a transaction with an offline Electrum wallet, using the Command line.

2.2.1 Create an unsigned transaction

With your online (watching-only) wallet, create an unsigned transaction:

```
electrum payto 1Cpf9zb5Rm5Z5qmmGezn6ERxFWvwuZ6UCx 0.1 --unsigned > unsigned.txn
```

The unsigned transaction is stored in a file named 'unsigned.txn'. Note that the –unsigned option is not needed if you use a watching-only wallet.

You may view it using:

```
cat unsigned.txn | electrum deserialize -
```

2.2.2 Sign the transaction

The serialization format of Electrum contains the master public key needed and key derivation, used by the offline wallet to sign the transaction.

Thus we only need to pass the serialized transaction to the offline wallet:

```
cat unsigned.txn | electrum signtransaction - > signed.txn
```

The command will ask for your password, and save the signed transaction in 'signed.txn'

2.2.3 Broadcast the transaction

Send your transaction to the Bitcoin network, using broadcast:

```
cat signed.txn | electrum broadcast -
```

If successful, the command will return the ID of the transaction.

2.3 How to accept Bitcoin on a website using Electrum

This tutorial will show you how to accept Bitcoin on a website with SSL signed payment requests. It is updated for Electrum 2.6.

2.3.1 Requirements

- A webserver serving static HTML
- A SSL certificate (signed by a CA)
- Electrum version >= 2.6

2.3.2 Create and use your merchant wallet

Create a wallet on your protected machine, as you want to keep your cryptocurrency safe. If anybody compromise your merchant server, s/he will be able to access read-only version of your wallet only and won't be able to spent currency.

Please notice that the potential intruder still will be able to see your addresses, transactions and balance, though. It's also recommended to use a separate wallet for your merchant purposes (and not your main wallet).

```
electrum create
```

Still being on a protected machine, export your Master Public Key (xpub):

```
electrum getmpk -w .electrum/wallets/your-wallet
```

Now you are able to set up your electrum merchant daemon.

On the server machine restore your wallet from previously exported Master Public Key (xpub):

```
electrum restore xpub...........................................
```

Once your read-only wallet is (re-)created, start Electrum as a daemon:

```
electrum daemon start
electrum daemon load_wallet
```

2.3.3 Add your SSL certificate to your configuration

You should have a private key and a public certificate for your domain.

Create a file that contains only the private key:

```
-----BEGIN PRIVATE KEY-----
your private key
-----END PRIVATE KEY-----
```

Set the path to your the private key file with setconfig:

```
electrum setconfig ssl_privkey /path/to/ssl.key
```

Create another file, file that contains your certificate, and the list of certificates it depends on, up to the root CA. Your certificate must be at the top of the list, and the root CA at the end.

```
-----BEGIN CERTIFICATE-----
your cert
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
intermediate cert
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
root cert
-----END CERTIFICATE-----
```

Set the ssl_chain path with setconfig:

```
electrum setconfig ssl_chain /path/to/ssl.chain
```

2.3.4 Configure a requests directory

This directory must be served by your webserver (eg Apache)

```
electrum setconfig requests_dir /var/www/r/
```

By default, electrum will display local URLs, starting with 'file://' In order to display public URLs, we need to set another configuration variable, url_rewrite. For example:

```
electrum setconfig url_rewrite "['file:///var/www/','https://electrum.org/']"
```

2.3.5 Create a signed payment request

```
electrum addrequest 3.14 -m "this is a test"
{
   "URI": "bitcoin:1MP49h5fbfLXiFpomsXeqJHGHUfNf3mCo4?amount=3.14&r=https://electrum.
↪org/r/7c2888541a",
   "address": "1MP49h5fbfLXiFpomsXeqJHGHUfNf3mCo4",
   "amount": 314000000,
   "amount (BTC)": "3.14",
   "exp": 3600,
   "id": "7c2888541a",
   "index_url": "https://electrum.org/r/index.html?id=7c2888541a",
   "memo": "this is a test",
   "request_url": "https://electrum.org/r/7c2888541a",
   "status": "Pending",
   "time": 1450175741
}
```

This command returns a json object with two URLs:

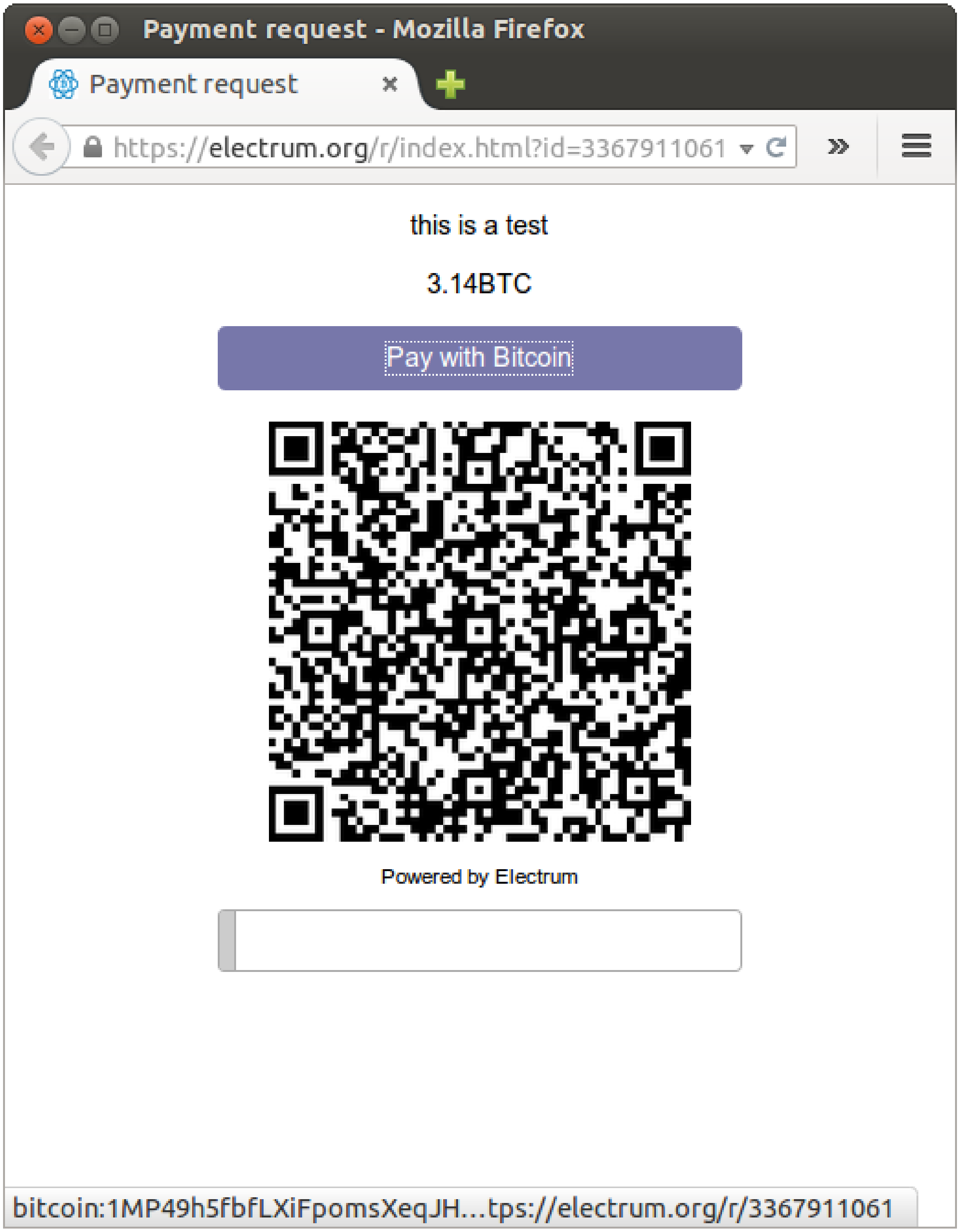
- request_url is the URL of the signed BIP70 request.
- index_url is the URL of a webpage displaying the request.

Note that request_url and index_url use the domain name we defined in url_rewrite.

You can view the current list of requests using the 'listrequests' command.

2.3.6 Open the payment request page in your browser

Let us open index_url in a web browser.

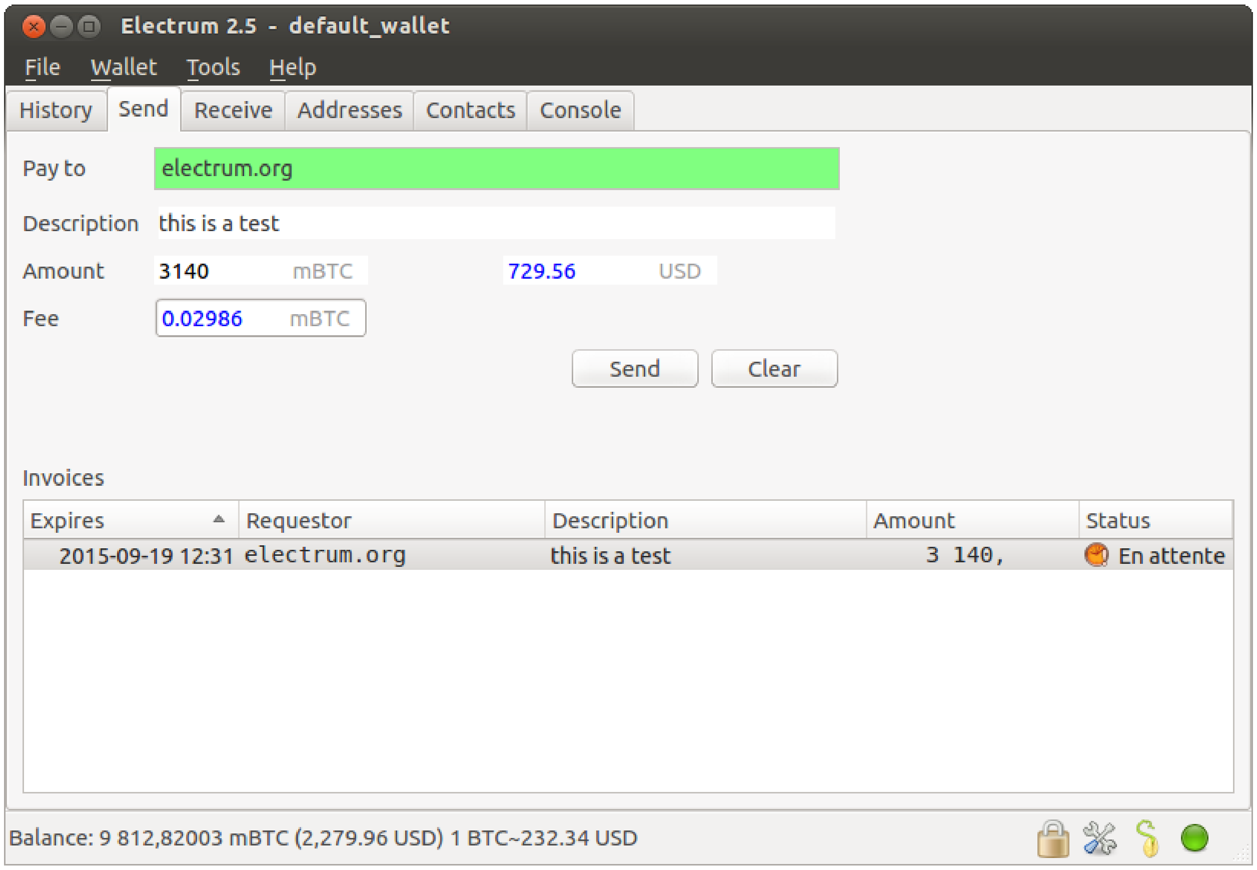

Payment request - Mozilla Firefox
Payment request
https://electrum.org/r/index.html?id=3367911061
Pay with Bitcoin
Powered by Electrum

this is a test

3.14BTC

bitcoin:1MP49h5fbfLXiFpomsXeqJH...tps://electrum.org/r/3367911061

The page shows the payment request. You can open the bitcoin: URI with a wallet, or scan the QR code. The bottom line displays the time remaining until the request expires.



Pay to: electrum.org

Description: this is a test

Amount: 3140 mBTC 729.56 USD

Fee: 0.02986 mBTC

Invoices

Expires	Requestor	Description	Amount	Status
2015-09-19 12:31	electrum.org	this is a test	3 140,	En attente

Balance: 9 812,82003 mBTC (2,279.96 USD) 1 BTC~232.34 USD

This page can already used to receive payments. However, it will not detect that a request has been paid; for that we need to configure websockets

2.3.7 Add web sockets support

Get SimpleWebSocketServer from here:

```
git clone https://github.com/ecdsa/simple-websocket-server.git
```

Set `websocket_server` and `websocket_port` in your config:

```
electrum setconfig websocket_server <FQDN of your server>

electrum setconfig websocket_port 9999
```

And restart the daemon:

```
electrum daemon stop

electrum daemon start
```

Now, the page is fully interactive: it will update itself when the payment is received. Please notice that higher ports might be blocked on some client's firewalls, so it is more safe for example to reverse proxy websockets transmission using standard `443` port on an additional subdomain.

2.3.8 JSONRPC interface

Commands to the Electrum daemon can be sent using JSONRPC. This is useful if you want to use electrum in a PHP script.

Note that the daemon uses a random port number by default. In order to use a stable port number, you need to set the 'rpcport' configuration variable (and to restart the daemon):

```
electrum setconfig rpcport 7777
```

Further, starting with Electrum 3.0.5, the JSON-RPC interface is authenticated using HTTP basic auth.

The username and the password are config variables. When first started, Electrum will initialise both; the password will be set to a random string. You can of course change them afterwards (the same way as the port, and then restart the daemon). To simply look up their value:

```
electrum getconfig rpcuser
electrum getconfig rpcpassword
```

Note that HTTP basic auth sends the username and the password unencrypted as part of the request. While using it on localhost is fine in our opinion, using it across an untrusted LAN or the Internet is not secure. Hence, you should take further measures in such cases, such as wrapping the connection in a secure tunnel. For further details, read this.

After setting a static port, and configuring authentication, we can perform queries using curl or PHP. Example:

```
curl --data-binary '{"id":"curltext","method":"getbalance","params":[]}' http://
↪username:password@127.0.0.1:7777
```

Query with named parameters:

```
curl --data-binary '{"id":"curltext","method":"listaddresses","params":{"funded":true}
↪}' http://username:password@127.0.0.1:7777
```

Create a payment request:

```
curl --data-binary '{"id":"curltext","method":"addrequest","params":{"amount":"3.14",
↪"memo":"test"}}' http://username:password@127.0.0.1:7777
```

2.4 How to split your coins using Electrum in case of a fork

2.4.1 Note:

This document has been updated for Electrum 2.9.

2.4.2 What is a fork?

A blockchain fork (or blockchain split) occurs when a deviating network begins to generate and maintain a conflicting chain of blocks branching from the original, essentially creating another "version of bitcoin" or cryptocurrency, with its very own blockchain, set of rules, and market value.

If there is a fork of the Bitcoin blockchain, two distinct currencies will coexist, having different market values.

2.4.3 What does it mean to 'split your coins'?

An address on the original blockchain will now also contain the same amount on the new chain.

If you own Bitcoins before the fork, a transaction that spends these coins after the fork will, in general, be valid on both chains. This means that you might be spending both coins simultaneously. This is called 'replay'. To prevent this, you need to move your coins using transactions that differ on both chains.

2.4.4 Fork detection

Electrum (version 2.9 and higher) is able to detect consensus failures between servers (blockchain forks), and lets users select their branch of the fork.

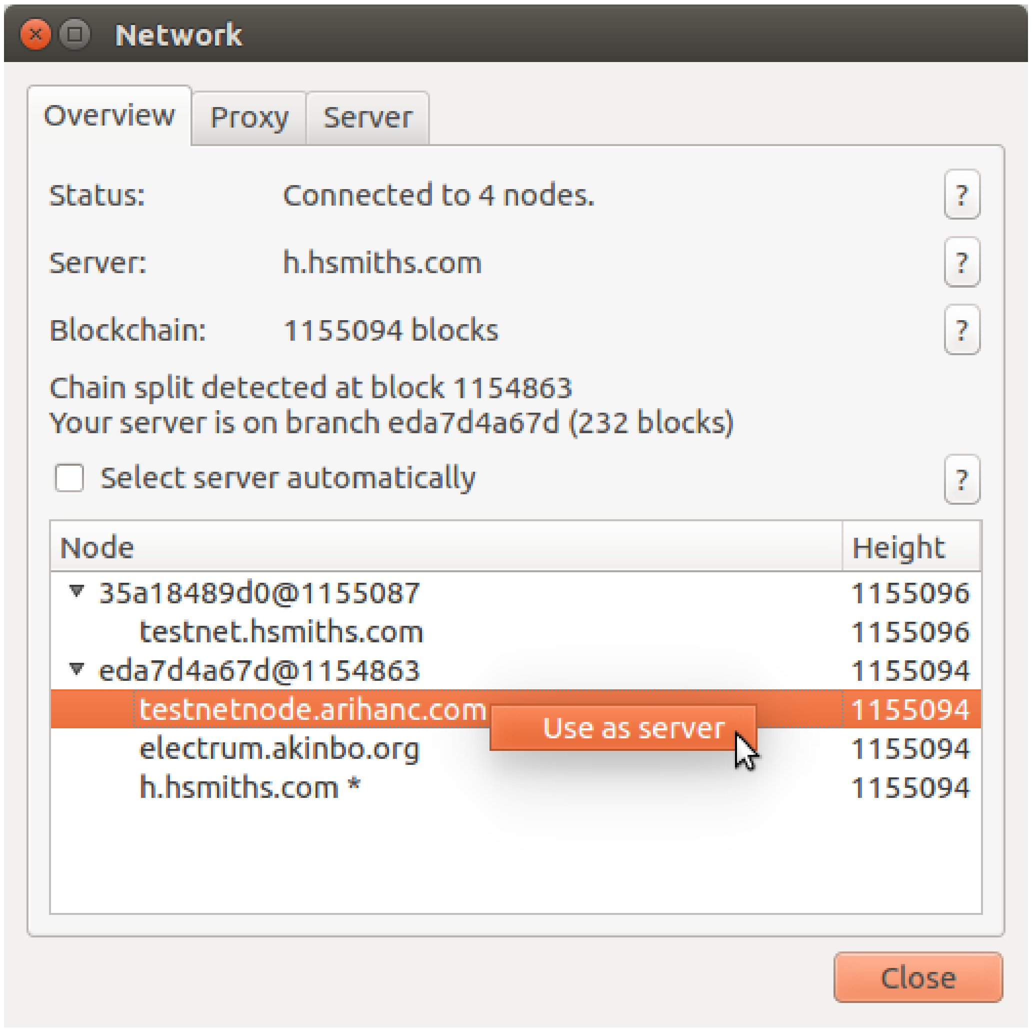
- Electrum will download and validate block headers sent by servers that may follow different branches of a fork in the Bitcoin blockchain. Instead of a linear sequence, block headers are organized in a tree structure. Branching points are located efficiently using binary search. The purpose of MCV is to detect and handle blockchain forks that are invisible to the classical SPV model.
- The desired branch of a blockchain fork can be selected using the network dialog. Branches are identified by the hash and height of the diverging block. Coin splitting is possible using RBF transaction (a tutorial will be added).

This feature allows you to pick and choose which chain and network you spend on.

2.4.5 Procedure

1. Preparation
   (a) Menu View Show Coins
   (b) Menu Tools Preferences Propose Replace-By-Fee "Always"
2. Select a chain / network
   (a) Menu Tools Network

   Notice how the branches have different hashes at different heights. You can verify which chain you're on by using block explorers to verify the hash and height.



Status: Connected to 4 nodes.

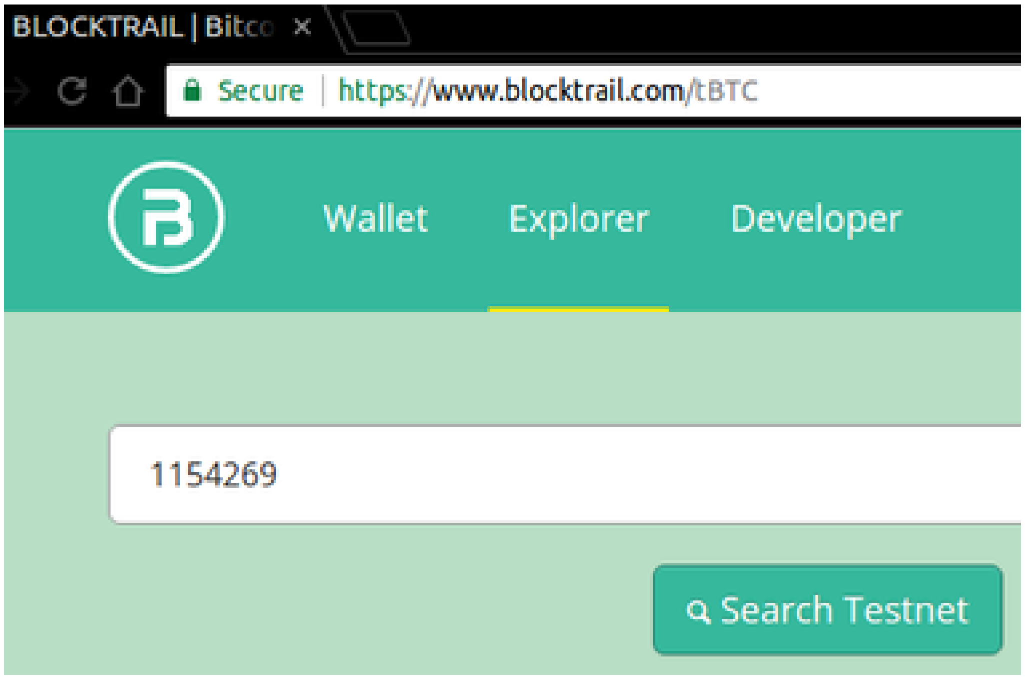
Server: h.hsmiths.com

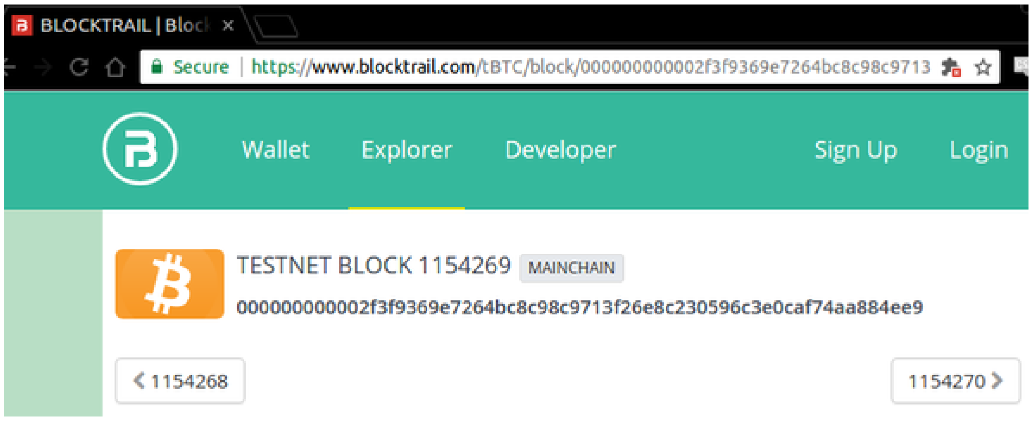
Blockchain: 1155094 blocks

Chain split detected at block 1154863
Your server is on branch eda7d4a67d (232 blocks)

☐ Select server automatically

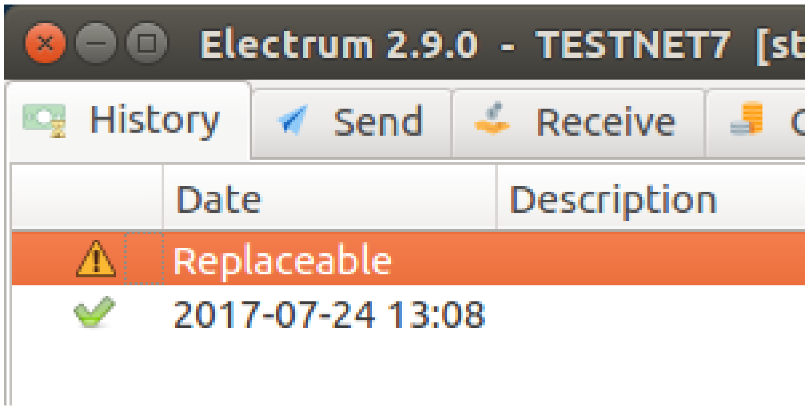
Node	Height
▼ 35a18489d0@1155087	1155096
testnet.hsmiths.com	1155096
▼ eda7d4a67d@1154863	1155094
testnetnode.arihanc.com	1155094
electrum.akinbo.org	1155094
h.hsmiths.com *	1155094



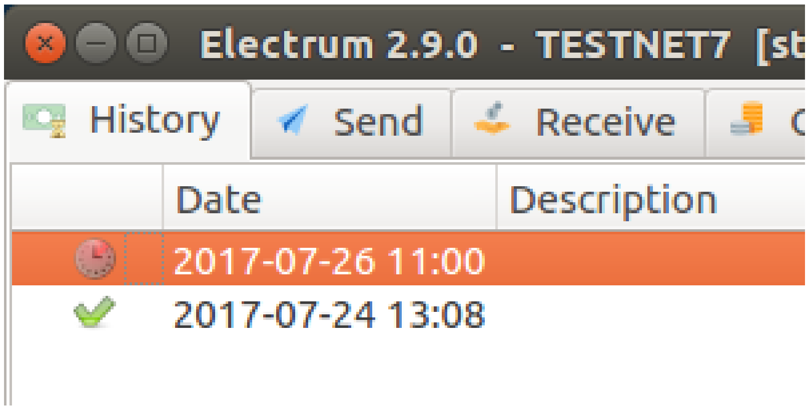

BLOCKTRAIL | Block
Secure | https://www.blocktrail.com/tBTC/block/000000000002f3f9369e7264bc8c98c9713
Wallet
Explorer
Developer
Sign Up
Login

TESTNET BLOCK 1154269 MAINCHAIN
000000000002f3f9369e7264bc8c98c9713f26e8c230596c3e0caf74aa884ee9

3. Send your coins to yourself
   (a) Copy your receiving address to the sending tab.
   (b) Enter how many coins you'd like to split. (enter " ! " for ALL)
   (c) Check "Replaceable"
   (d) Send Sign Broadcast
4. Wait for the transaction to confirm on one network.
   (a) You'll want to switch between chains (via the network panel) to monitor the transaction status.
   (b) Wait until you see the transaction confirm on one chain.

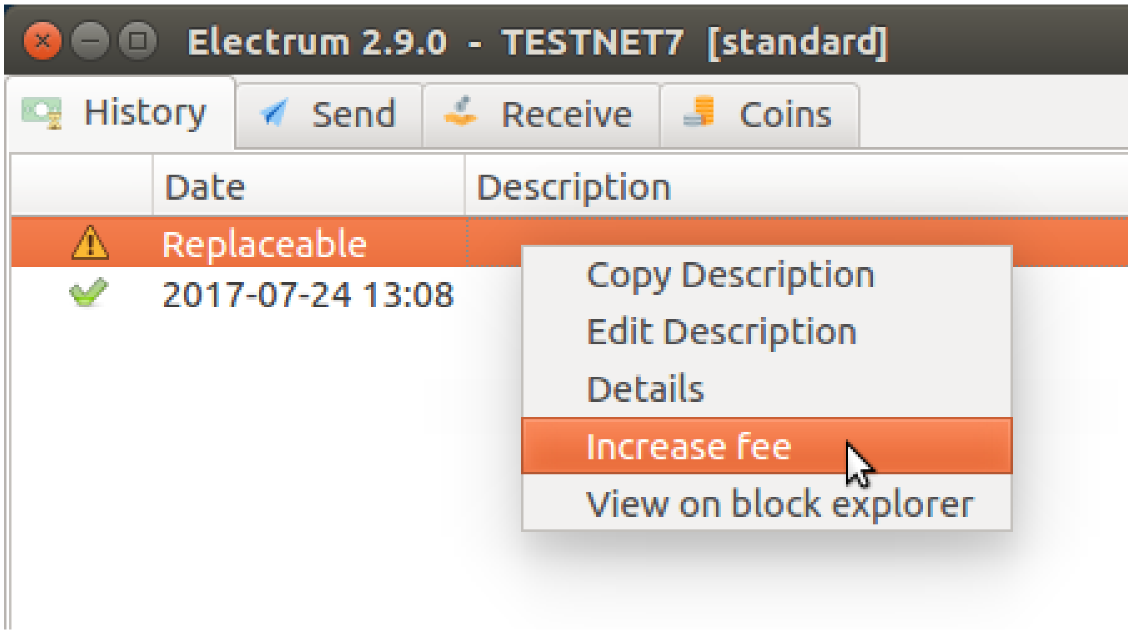


	Date	Description
⚠	Replaceable	
✓	2017-07-24 13:08	

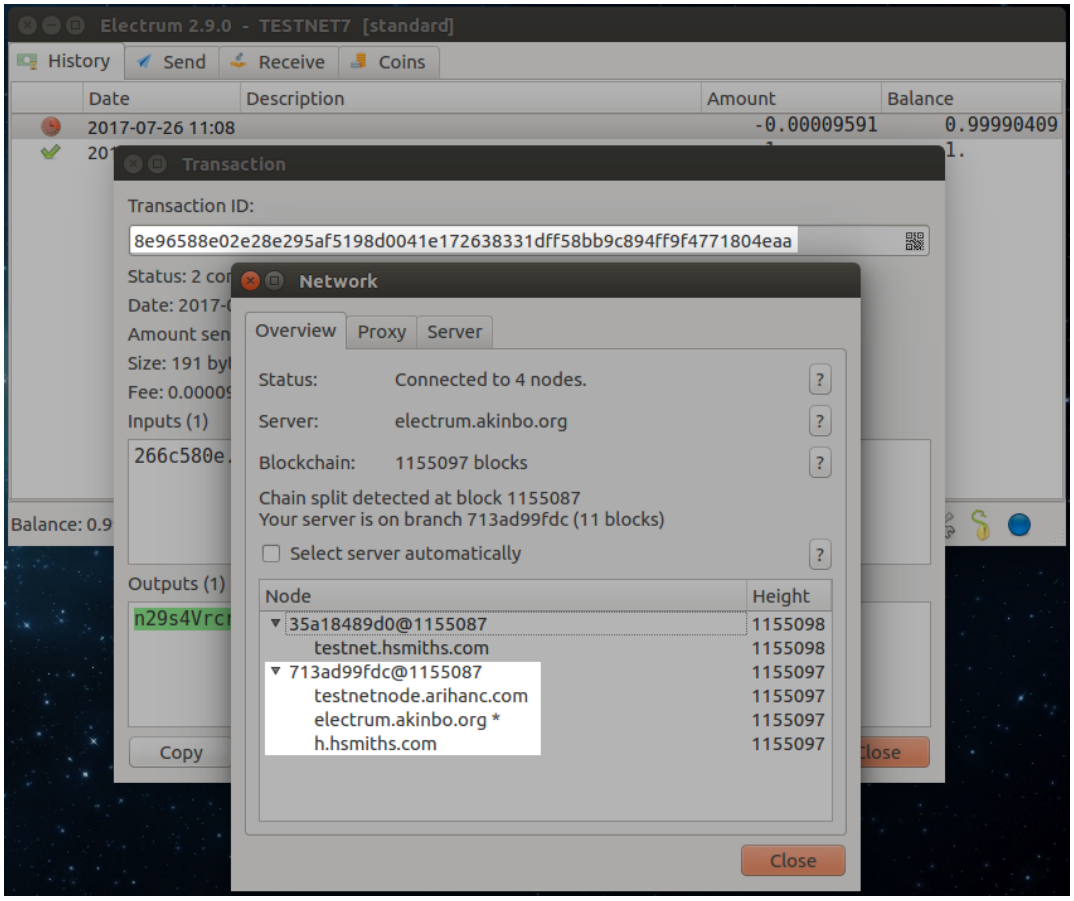


	Date	Description
🕒	2017-07-26 11:00	
✓	2017-07-24 13:08	

   (c) Immediately use "RBF" on the unconfirmed transaction to "Increase fee"



5. Wait for both chains to confirm the transaction.
6. Verify the transaction has a different TXID on each chain.



Date	Description	Amount	Balance
2017-07-26 11:08		-0.00009591	0.99990409
20			1.

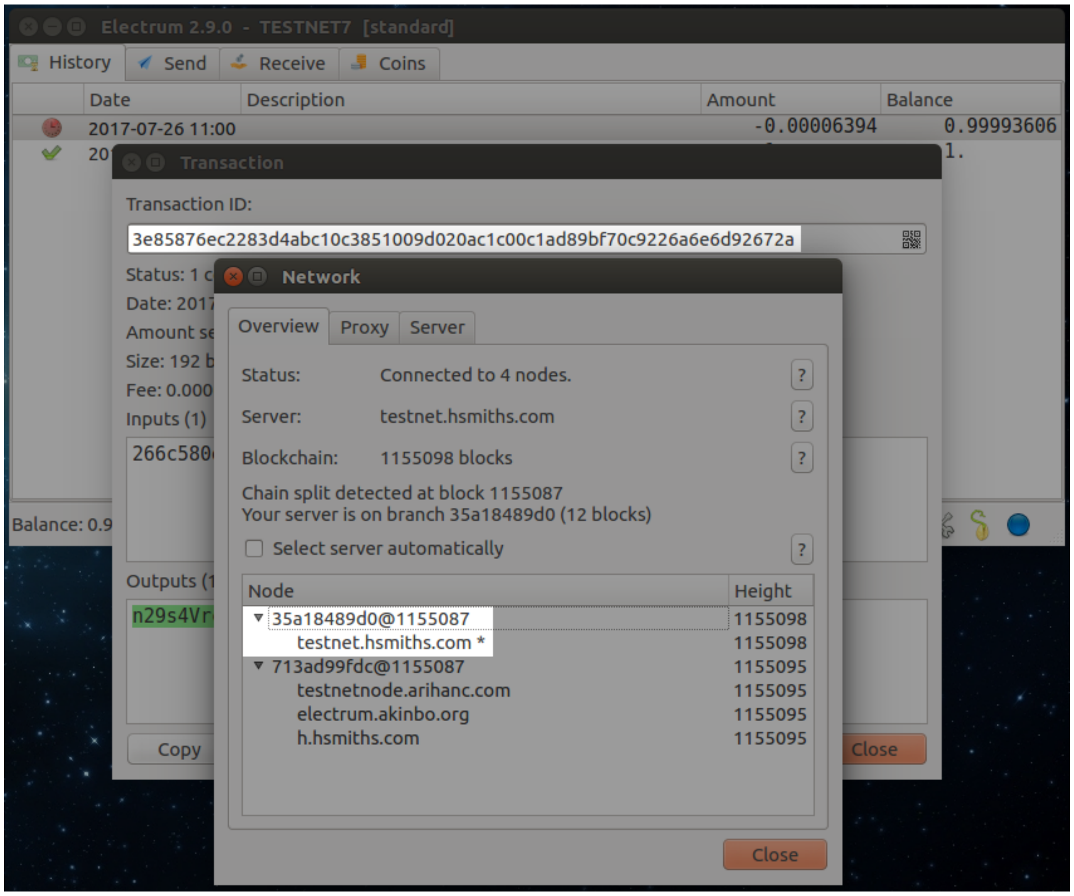
Balance: 0.9

Transaction ID:
8e96588e02e28e295af5198d0041e172638331dff58bb9c894ff9f4771804eaa

Status: 2 co
Date: 2017-
Amount sen
Size: 191 by
Fee: 0.0000
Inputs (1)
266c580e.
Outputs (1)
n29s4Vrc

Status: Connected to 4 nodes.
Server: electrum.akinbo.org
Blockchain: 1155097 blocks
Chain split detected at block 1155087
Your server is on branch 713ad99fdc (11 blocks)
☐ Select server automatically

Node	Height
▼ 35a18489d0@1155087	1155098
testnet.hsmiths.com	1155098
▼ 713ad99fdc@1155087	1155097
testnetnode.arihanc.com	1155097
electrum.akinbo.org *	1155097
h.hsmiths.com	1155097



Transaction ID:
3e85876ec2283d4abc10c3851009d020ac1c00c1ad89bf70c9226a6e6d92672a

Status: Connected to 4 nodes.

Server: testnet.hsmiths.com

Blockchain: 1155098 blocks

Chain split detected at block 1155087
Your server is on branch 35a18489d0 (12 blocks)

Select server automatically

```
Node                          Height
▾ 35a18489d0@1155087          1155098
    testnet.hsmiths.com *     1155098
▾ 713ad99fdc@1155087          1155095
    testnetnode.arihanc.com   1155095
    electrum.akinbo.org       1155095
    h.hsmiths.com             1155095
```

You will now have coins seperately spendable on each chain. If it failed, no harm done, you sent to yourself! Just try again.

2.5 Using Electrum Through Tor

Please note that when using electrum through tor there are two main ways. The first way has the most Privacy but also requires the most trust in the server you are connecting too. This is because normally Electrum connects to a few different servers and downloads block headers and checks that they match. This prevents / makes it more difficult for Rogue servers to send you bad information. However this can also present a privacy issue because you could be connecting to none .onion servers for these headers.

Thus the two different options are, Connect to 1 server ONLY and get block headers and transaction info from that server. Or Connect to 8 block header servers and connect to 1 .onion server for the general use.

2.5.1 List of Active .onion servers

Check the list here;

http://electrumserv.noip.me/onionservers.txt

If you wish to be added to this list email me at:

danielcryptos@gmail.com

2.5.2 LINUX

2.5.3 Option 1: Single Server

Note: Please understand you are sacrificing some security here for extra privacy.

Check out https://electrum.org/#download

Grab the download from python source

Make sure you have dependencies installed

sudo apt-get install python-qt4 python-pip

Extract the electrum download;

tar -xvzf Electrum-2.*.*.tar.gz

Go into the extracted electrum folder and then run;

./electrum -1 -s electrums3lojbuj.onion:50001:t -p socks5:localhost:9050

Quick explanation,

-1 means connect to 1 server only.

-s is defining which server. You can change this to any .onion server you want. (Check the list at the top)

-p Is saying what proxy server to use to get into the tor network. Generally this will be localhost but the port bit after : could be different.

You might need to change the port bit depending on what system you are running;

Currently The port is;

Tor Browser Bundle: 9150

General Tor (Installed): 9050

2.5.4 Option 2: Multiple servers but Tor Main

Same as above until the command to launch electrum, Remove the -1 making it

./electrum -s electrums3lojbuj.onion:50001:t -p socks5:localhost:9050

For this one you can also just launch electrum and click on the Green or Red icon on the bottom right to bring up server information

Untick the box for Auto and enter;

electrums3lojbuj.onion

50001

Into the boxes.

At the bottom select SOCKS5 for proxy and then

localhost

9150 or 9050

2.5.5 WINDOWS

2.5.6 Option 1: Connecting to a single Server

Install electrum from the main download page; https://electrum.org/#download

Note: Please understand you are sacrificing some security here for extra privacy.

In windows, On your desktop you will have a electrum icon. Copy and paste this to make a copy. If not you can find the electrum folder in C:Program Files (x86)Electrum

Right click on electrum.exe and create shortcut. It will say cannot make a shortcut here make one on the desktop instead? Ok this.

With your new shortcut or a copy of your old one Right click it and go properties, click shortcut at the top bar, in the box named target:

It should already say something similar to what's in between the speech bubbles. If yours is different don't change that bit to match.

What we want to do is add on the bit after the last speech bubble. Make a space and then enter / copy and paste.

“C:Program Files (x86)Electrumelectrum.exe” -1 -s electrums3lojbuj.onion:50001:t -p socks5:localhost:9050

Apply and Ok the change… You can go back to the General Tab if you want and Where it says “electrum.exe - Shortcut” you could change that to Electrum - Tor or something

Click apply and ok again.

Now when you launch Electrum with this shortcut it will use 1 tor server only.

Quick explanation,

-1 means connect to 1 server only.

-s is defining which server. You can change this to any .onion server you want.

-p Is saying what proxy server to use to get into the tor network. Generally this will be localhost but the port bit after : could be different.

You might need to change the port bit depending on what system you are running;

Currently The port is;

Tor Browser Bundle: 9150

General Tor (Installed): 9050

2.5.7 Option 2

In windows, On your desktop you will have a electrum icon. Copy and paste this to make a copy. If not you can find the electrum folder in C:Program Files (x86)Electrum

Right click on electrum.exe and create shortcut. It will say cannot make a shortcut here make one on the desktop instead? Ok this.

With your new shortcut or a copy of your old one Right click it and go properties, click shortcut at the top bar, in the box named target:

It should already say something similar to what's in between the speech bubbles. If yours is different don't change that bit to match.

What we want to do is add on the bit after the last speech bubble. Make a space and then enter / copy and paste.

"C:Program Files (x86)Electrumelectrum.exe" -s electrums3lojbuj.onion:50001:t -p socks5:localhost:9050

Apply and Ok the change… You can go back to the General Tab if you want and Where it says "electrum.exe - Shortcut" you could change that to Electrum - Tor or something

Click apply and ok again.

Now when you launch Electrum with this shortcut it will use 1 tor server only.

You might need to change the port bit depending on what system you are running;

Currently The port is;

Tor Browser Bundle: 9150

General Tor (Installed): 9050

For this one you can also just launch electrum and click on the Green or Red icon on the bottom right to bring up server information Untick the box for Auto and enter;

electrums3lojbuj.onion

50001

Into the boxes.

At the bottom select SOCKS5 for proxy and then

localhost

9150 or 9050

CHAPTER 3

For developers

3.1 The Python Console

Most Electrum commands are available not only using the command-line, but also in the GUI Python console.

The results are Python objects, even though they are sometimes rendered as JSON for clarity.

Let us call listunspent(), to see the list of unspent outputs in the wallet:

```
>> listunspent()
[
 {
     "address": "12cmY5RHRgx8KkUKASDcDYRotget9FNso3",
     "index": 0,
     "raw_output_script": "76a91411bbdc6e3a27c44644d83f783ca7df3bdc2778e688ac",
     "tx_hash": "e7029df9ac8735b04e8e957d0ce73987b5c9c5e920ec4a445130cdeca654f096",
     "value": 0.01
 },
 {
     "address": "1GavSCND6TB7HuCnJSTEbHEmCctNGeJwXF",
     "index": 0,
     "raw_output_script": "76a914aaf437e25805f288141bfcdc27887ee5492bd13188ac",
     "tx_hash": "b30edf57ca2a31560b5b6e8dfe567734eb9f7d3259bb334653276efe520735df",
     "value": 9.04735316
 }
]
```

Note that the result is rendered as JSON. However, if we save it to a Python variable, it is rendered as a Python object:

```
>> u = listunspent()
>> u
[{'tx_hash': u'e7029df9ac8735b04e8e957d0ce73987b5c9c5e920ec4a445130cdeca654f096',
↪'index': 0, 'raw_output_script': '76a91411bbdc6e3a27c44644d83f783ca7df3bdc2778e688ac
↪', 'value': 0.01, 'address': '12cmY5RHRgx8KkUKASDcDYRotget9FNso3'}, {'tx_hash': u
↪'b30edf57ca2a31560b5b6e8dfe567734eb9f7d3259bb334653276efe520735df', 'index': 0,
↪'raw_output_script': '76a914aaf437e25805f288141bfcdc27887ee5492bd13188ac', 'value':␣
↪9.04735316, 'address': '1GavSCND6TB7HuCnJSTEbHEmCctNGeJwXF'}]
```

This makes it possible to combine Electrum commands with Python. For example, let us pick only the addresses in the previous result:

```
>> map(lambda x:x.get('address'), listunspent())
[
 "12cmY5RHRgx8KkUKASDcDYRotget9FNso3",
 "1GavSCND6TB7HuCnJSTEbHEmCctNGeJwXF"
]
```

Here we combine two commands, listunspent and dumpprivkeys, in order to dump the private keys of all adresses that have unspent outputs:

```
>> dumpprivkeys( map(lambda x:x.get('address'), listunspent()) )
{
 "12cmY5RHRgx8KkUKASDcDYRotget9FNso3":
↪"***************************************************",
 "1GavSCND6TB7HuCnJSTEbHEmCctNGeJwXF":
↪"***************************************************"
}
```

Note that dumpprivkey will ask for your password if your wallet is encrypted. The GUI methods can be accessed through the gui variable. For example, you can display a QR code from a string using gui.show_qrcode. Example:

```
gui.show_qrcode(dumpprivkey(listunspent()[0]['address']))
```

3.2 Simple Payment Verification

Simple Payment Verification (SPV) is a technique described in Satoshi Nakamoto's paper. SPV allows a lightweight client to verify that a transaction is included in the Bitcoin blockchain, without downloading the entire blockchain. The SPV client only needs download the block headers, which are much smaller than the full blocks. To verify that a transaction is in a block, a SPV client requests a proof of inclusion, in the form of a Merkle branch.

SPV clients offer more security than web wallets, because they do not need to trust the servers with the information they send.

Reference: Bitcoin: A peer-to-peer Electronic Cash System

3.3 Electrum Seed Version System

This document describes the Seed Version System used in Electrum (version 2.0 and higher).

3.3.1 Description

Electrum derives its private keys and addresses from a seed phrase made of natural language words. Starting with version 2.0, Electrum seed phrases include a version number, whose purpose is to indicate which derivation should be followed in order to derive private keys and addresses.

In order to eliminate the dependency on a fixed wordlist, the master private key and the version number are both obtained by hashes of the UTF8 normalized seed phrase. The version number is obtained by looking at the first bits of:

```
hmac_sha_512("Seed version", seed_phrase)
```

The version number is also used to check seed integrity; in order to be correct, a seed phrase must produce a registered version number.

3.3.2 Motivation

Early versions of Electrum (before 2.0) used a bidirectional encoding between seed phrase and entropy. This type of encoding requires a fixed wordlist. This means that future versions of Electrum must ship with the exact same wordlist, in order to be able to read old seed phrases.

BIP39 was introduced two years after Electrum. BIP39 seeds include a checksum, in order to help users figure out typing errors. However, BIP39 suffers the same shortcomings as early Electrum seed phrases:

- A fixed wordlist is still required. Following our recommendation, BIP39 authors decided to derive keys and addresses in a way that does not depend on the wordlist. However, BIP39 still requires the wordlist in order to compute its checksum, which is plainly inconsistent, and defeats the purpose of our recommendation. This problem is exacerbated by the fact that BIP39 proposes to create one wordlist per language. This threatens the portability of BIP39 seed phrases.
- BIP39 seed phrases do not include a version number. This means that software should always know how to generate keys and addresses. BIP43 suggests that wallet software will try various existing derivation schemes within the BIP32 framework. This is extremely inefficient and rests on the assumption that future wallets will support all previously accepted derivation methods. If, in the future, a wallet developer decides not to implement a particular derivation method because it is deprecated, then the software will not be able to detect that the corresponding seed phrases are not supported, and it will return an empty wallet instead. This threatens users funds.

For these reasons, Electrum does not generate BIP39 seeds. Starting with version 2.0, Electrum uses the following Seed Version System, which addresses these issues.

Electrum 2.0 derives keys and addresses from a hash of the UTF8 normalized seed phrase with no dependency on a fixed wordlist. This means that the wordlist can differ between wallets while the seed remains portable, and that future wallet implementations will not need today's wordlists in order to be able to decode the seeds created today. This reduces the cost of forward compatibility.

3.3.3 Version number

The version number is a prefix of a hash derived from the seed phrase. The length of the prefix is a multiple of 4 bits. The prefix is computed as follows:

```
def version_number(seed_phrase):
  # normalize seed
  normalized = prepare_seed(seed_phrase)
  # compute hash
  h = hmac_sha_512("Seed version", normalized)
  # use hex encoding, because prefix length is a multiple of 4 bits
  s = h.encode('hex')
  # the length of the prefix is written on the fist 4 bits
  # for example, the prefix '101' is of length 4*3 bits = 4*(1+2)
  length = int(s[0]) + 2
  # read the prefix
  prefix = s[0:length]
  # return version number
  return hex(int(prefix, 16))
```

The normalization function (prepare_seed) removes all but one space between words. It also removes diacritics, and it removes spaces between Asian CJK characters.

3.3.4 List of reserved numbers

The following version numbers are used for Electrum seeds.

Number	Type	Description
0x01	Standard	P2PKH and Multisig P2SH wallets
0x100	Segwit	Segwit: P2WPKH and P2WSH wallets
0x101	2FA	Two-factor authenticated wallets

In addition, the version bytes of master public/private keys indicate what type of output script should be used. The following prefixes are used for master public keys:

Version	Prefix	Description
0x0488b21e	xpub	P2PKH or P2SH
0x049d7cb2	ypub	P2WPKH in P2SH
0x0295b43f	Ypub	P2WSH in P2SH
0x04b24746	zpub	P2WPKH
0x02aa7ed3	Zpub	P2WSH

And for master private keys:

Version	Prefix	Description
0x0488ade4	xprv	P2PKH or P2SH
0x049d7878	yprv	P2WPKH in P2SH
0x0295b005	Yprv	P2WSH in P2SH
0x04b2430c	zprv	P2WPKH
0x02aa7a99	Zprv	P2WSH

3.3.5 Seed generation

When the seed phrase is hashed during seed generation, the resulting hash must begin with the correct version number prefix. This is achieved by enumerating a nonce and re-hashing the seed phrase until the desired version number is created. This requirement does not decrease the security of the seed (up to the cost of key stretching, that might be required to generate the private keys).

3.3.6 Security implications

Electrum currently use the same wordlist as BIP39 (2048 words). A typical seed has 12 words, which results in 132 bits of entropy in the choice of the seed.

Following BIP39, 2048 iterations of key stretching are added for the generation of the master private key. In terms of hashes, this is equivalent to adding an extra 11 bits of security to the seed (2048=2^11).

From the point of view of an attacker, the constraint added by imposing a prefix to the seed version hash does not decrease the entropy of the seed, because there is no knowledge gained on the seed phrase. The attacker still needs to enumerate and test 2^n candidate seed phrases, where n is the number of bits of entropy used to generate the seed.

However, the test made by the attacker will return faster if the candidate seed is not a valid seed, because the attacker does not need to generate the key. This means that the imposed prefix reduces the strength of key stretching.

Let n denote the number of entropy bits of the seed, and m the number of bits of difficulty added by key stretching: m = log2(stretching_iterations). Let k denote the length of the prefix, in bits.

On each iteration of the attack, the probability to obtain a valid seed is p = 2^-k

The number of hashes required to test a candidate seed is: p * (1+2^m) + (1-p)*1 = 1 + 2^(m-k)

Therefore, the cost of an attack is: 2^n * (1 + 2^(m-k))

This can be approximated as 2^(n + m - k) if m>k and as 2^n otherwise.

With the standard values currently used in Electrum, we obtain: 2^(132 + 11 - 8) = 2^135. This means that a standard Electrum seed is equivalent, in terms of hashes, to 135 bits of entropy.

3.4 Electrum protocol specification

Stratum is a universal bitcoin communication protocol used mainly by bitcoin client Electrum and miners.

3.4.1 Format

Stratum protocol is based on JSON-RPC 2.0 (although it doesn't include "jsonrpc" information in every message). Each message has to end with a line end character (n).

Request

Typical request looks like this:

```
{ "id": 0, "method":"some.stratum.method", "params": [] }
```

- id begins at 0 and every message has its unique id number
- list and description of possible methods is below
- params is an array, e.g.: ["1myfirstaddress", "1mysecondaddress", "1andonemoreaddress"]

Response

Responses are similar:

```
{ "id": 0, "result": "616be06545e5dd7daec52338858b6674d29ee6234ff1d50120f060f79630543c
↪"}
```

- id is copied from the request message (this way client can pair each response to one of his requests)
- result can be:
  - null
  - a string (as shown above)
  - a hash, e.g.:

```
{ "nonce": 1122273605, "timestamp": 1407651121, "version": 2, "bits":␣
↪406305378 }
```

- an array of hashes, e.g.:

```
[ { "tx_hash:
"b87bc42725143f37558a0b41a664786d9e991ba89d43a53844ed7b3752545d4f",
"height": 314847 }, { "tx_hash":
"616be06545e5dd7daec52338858b6674d29ee6234ff1d50120f060f79630543c",
"height": 314853 } ]
```

3.4.2 Methods

server.version

This is usually the first client's message, plus it's sent every minute as a keep-alive message. Client sends its own version and version of the protocol it supports. Server responds with its supported version of the protocol (higher number at server-side is usually compatible).

The version of the protocol being explained in this documentation is: 0.10.

request:

```
{ "id": 0, "method": "server.version", "params": [ "1.9.5", "0.6" ] }
```

response:

```
{ "id": 0, "result": "0.8" }
```

server.banner

request:

```
{ "id": 1, "method": "server.banner", "params": [] }
```

server.donation_address

server.peers.subscribe

Client can this way ask for a list of other active servers. Servers are connected to an IRC channel (#electrum at freenode.net) where they can see each other. Each server announces its version, history pruning limit of every address ("p100", "p10000" etc.–the number means how many transactions the server may keep for every single address) and supported protocols ("t" = tcp@50001, "h" = http@8081, "s" = tcp/tls@50002, "g" = https@8082; non-standard port would be announced this way: "t3300" for tcp on port 3300).

Note: At the time of writing there isn't a true subscription implementation of this method, but servers only send one-time response. They don't send notifications yet.

request:

```
{ "id": 3, "method":
"server.peers.subscribe", "params": [] }<br/>
```

response:

```
{ "id": 3, "result": [ [ "83.212.111.114",
"electrum.stepkrav.pw", [ "v0.9", "p100", "t", "h", "s",
"g" ] ], [ "23.94.27.149", "ultra-feather.net", [ "v0.9",
"p10000", "t", "h", "s", "g" ] ], [ "88.198.241.196",
"electrum.be", [ "v0.9", "p10000", "t", "h", "s", "g" ] ] ]
}
```

blockchain.numblocks.subscribe

A request to send to the client notifications about new blocks height. Responds with the current block height.

request:

```
{ "id": 5, "method":
"blockchain.numblocks.subscribe", "params": [] }
```

response:

```
{ "id": 5, "result": 316024 }
```

message:

```
{ "id": null, "method":
"blockchain.numblocks.subscribe", "params": 316024 }
```

blockchain.headers.subscribe

A request to send to the client notifications about new blocks in form of parsed blockheaders.

request:

```
{ "id": 5, "method":
"blockchain.headers.subscribe", "params": [] }
```

response:

```
{ "id": 5, "result": { "nonce":
3355909169, "prev_block_hash":
"00000000000000002b3ef284c2c754ab6e6abc40a0e31a974f966d8a2b4d5206",
"timestamp": 1408252887, "merkle_root":
"6d979a3d8d0f8757ed96adcd4781b9707cc192824e398679833abcb2afdf8d73",
"block_height": 316023, "utxo_root":
"4220a1a3ed99d2621c397c742e81c95be054c81078d7eeb34736e2cdd7506a03",
"version": 2, "bits": 406305378 } }
```

message:

```
{ "id": null, "method":
"blockchain.headers.subscribe", "params": [ { "nonce":
881881510, "prev_block_hash":
"00000000000000001ba892b1717690900ae476857120a78fb50825f8b67a42d4",
"timestamp": 1408255430, "merkle_root":
"8e92bdbf1c5c581b5942fc290c6c52c586f091b279ea79d4e21460e138023839",
"block_height": 316024, "utxo_root":
"060f780c0dd07c4289aaaa2ef24723f73380095b31d60795e1308170ec742ffb",
"version": 2, "bits": 406305378 } ] }
```

blockchain.address.subscribe

A request to send to the client notifications when status (i.e., transaction history) of the given address changes. Status is a hash of the transaction history. If there isn't any transaction for the address yet, the status is null.

request:

```
{ "id": 6, "method":"blockchain.address.subscribe", "params": [
↪"1NS17iag9jJgTHD1VXjvLCEnZuQ3rJDE9L"] }
```

response:

```
{ "id": 6, "result":"b87bc42725143f37558a0b41a664786d9e991ba89d43a53844ed7b3752545d4f
↪" }
```

message:

```
{ "id": null, "method":"blockchain.address.subscribe", "params": [
↪"1NS17iag9jJgTHD1VXjvLCEnZuQ3rJDE9L",
↪"690ce08a148447f482eb3a74d714f30a6d4fe06a918a0893d823fd4aca4df580"]}
```

blockchain.address.get_history

For a given address a list of transactions and their heights (and fees in newer versions) is returned.

request:

```
{"id": 1, "method": "blockchain.address.get_history", "params": [
↪"1NS17iag9jJgTHD1VXjvLCEnZuQ3rJDE9L"] }
```

response:

```
{"id": 1, "result": [{"tx_hash":
↪"ac9cd2f02ac3423b022e86708b66aa456a7c863b9730f7ce5bc24066031fdced", "height":␣
↪340235}, {"tx_hash":
↪"c4a86b1324f0a1217c80829e9209900bc1862beb23e618f1be4404145baa5ef3", "height":␣
↪340237}]}
{"jsonrpc": "2.0", "id": 1, "result": [{"tx_hash":
↪"16c2976eccd2b6fc937d24a3a9f3477b88a18b2c0cdbe58c40ee774b5291a0fe", "height": 0,
↪"fee": 225}]}
```

blockchain.address.get_mempool

blockchain.address.get_balance

request:

```
{ "id": 1, "method":"blockchain.address.get_balance", "params":[
↪"1NS17iag9jJgTHD1VXjvLCEnZuQ3rJDE9L"] }
```

response:

```
{"id": 1, "result": {"confirmed": 533506535, "unconfirmed": 27060000}}
```

blockchain.address.get_proof

blockchain.address.listunspent

request:

```
{ "id": 1, "method":
"blockchain.address.listunspent", "params":
["1NS17iag9jJgTHD1VXjvLCEnZuQ3rJDE9L"] }<br/>
```

response:

```
{"id": 1, "result": [{"tx_hash":
"561534ec392fa8eebf5779b233232f7f7df5fd5179c3c640d84378ee6274686b",
"tx_pos": 0, "value": 24990000, "height": 340242},
{"tx_hash":"620238ab90af02713f3aef314f68c1d695bbc2e9652b38c31c025d58ec3ba968",
"tx_pos": 1, "value": 19890000, "height": 340242}]}
```

blockchain.utxo.get_address

blockchain.block.get_header

blockchain.block.get_chunk

blockchain.transaction.broadcast

Submits raw transaction (serialized, hex-encoded) to the network. Returns transaction id, or an error if the transaction is invalid for any reason.

request:

```
{ "id": 1, "method":
"blockchain.transaction.broadcast", "params":

↪"0100000002f327e86da3e66bd20e1129b1fb36d07056f0b9a117199e759396526b8f3a20780000000000fffffffff0ede0
↪" }<br/>
```

response:

```
{"id": 1, "result": "561534ec392fa8eebf5779b233232f7f7df5fd5179c3c640d84378ee6274686b
↪"}
```

blockchain.transaction.get_merkle

> blockchain.transaction.get_merkle [$txid, $txHeight]

blockchain.transaction.get

Method for obtaining raw transaction (hex-encoded) for given txid. If the transaction doesn't exist, an error is returned.

request:

```
{ "id": 17, "method":"blockchain.transaction.get", "params": [
"0e3e2357e806b6cdb1f70b54c3a3a17b6714ee1f0e68bebb44a74b1efd512098"
] }
```

response:

```
{ "id": 17, "result":
↪"01000000010000000000000000000000000000000000000000000000000000000000000000ffffffff0704ffff001d0104
↪"}
```

error:

```
{ "id": 17, "error": "{ u'message': u'No information available about transaction', u
↪'code': -5 }" }
```

blockchain.estimatefee

Estimates the transaction fee per kilobyte that needs to be paid for a transaction to be included within a certain number of blocks. If the node doesn't have enough information to make an estimate, the value -1 will be returned.

Parameter: How many blocks the transaction may wait before being included.

request:

```
{ "id": 17, "method": "blockchain.estimatefee", "params": [ 6 ] }
```

response:

```
{ "id": 17, "result": 0.00026809 }
{ "id": 17, "result": 1.169e-05 }
```

error:

```
{ "id": 17, "result": -1 }
```

3.4.3 External links

- https://docs.google.com/a/palatinus.cz/document/d/17zHy1SUlhgtCMbypO8cHgpWH73V5iUQKk_0rWvMqSNs/edit?hl=en_US" original Slush's specification of Stratum protocol
- http://mining.bitcoin.cz/stratum-mining specification of Stratum mining extension

3.5 Serialization of unsigned or partially signed transactions

Electrum 2.0 uses an extended serialization format for transactions. The purpose of this format is to send unsigned and partially signed transactions to cosigners or to cold storage.

This is achieved by extending the 'pubkey' field of a transaction input.

3.5.1 Extended public keys

The first byte of the pubkey indicates if it is an extended pubkey:

- 0x02, 0x03, 0x04: legal Bitcoin public key (compressed or not).
- 0xFF, 0xFE, 0xFD: extended public key.

Extended public keys are of 3 types:

- 0xFF: bip32 xpub and derivation
- 0xFE: legacy electrum derivation: master public key + derivation
- 0xFD: unknown pubkey, but we know the Bitcoin address.

3.5.2 Public key

This is the legit Bitcoin serialization of public keys.

0x02 or 0x03	compressed public key (32 bytes)
0x04	uncompressed public key (64 bytes)

3.5.3 BIP32 derivation

0xFF	xpub (78 bytes)	bip32 derivation (2*k bytes)

3.5.4 Legacy Electrum Derivation

0xFE	mpk (64 bytes)	derivation (4 bytes)

3.5.5 Bitcoin address

Used if we don't know the public key, but we know the address (or the hash 160 of the output script). The cosigner should know the public key.

0xFD	hash_160_of_script (20 bytes)