ElectrumSV

The ElectrumSV developers

Oct 08, 2020

GETTING STARTED

ElectrumSV is a wallet application for Bitcoin SV, a peer to peer form of electronic cash. As a wallet application it allows you to track, receive and spend bitcoin whenever you need to. But that's just the basics, as it manages and secures your keys it also helps you to do many other things.

Important: ElectrumSV can only be downloaded from electrumsv.io.

CHAPTER

ONE

GETTING STARTED

Before you can send and receive payments, you need to first create a wallet, and then create at least one account within it.

How do you create a wallet? Your wallet is a standalone container for all your bitcoin-related data. You should be able to create as many accounts as you need within it, each account containing separated funds much like a bank account. Read more about *creating a wallet*.

How do you create an account? Each account in your wallet is much like a bank account, with the funds in each separated from the others. Read more about *creating an account*.

How do you receive a payment from someone else? Each account has the ability to provide countless unique and private receiving addresses and by giving a different one of these out to each person who will send you coins, allows you to receive funds from them. Read more about *receiving a payment*.

How do you make a payment to someone else? By obtaining an address from another person, if you have coins in one of your accounts, you should be able to send some or all of those coins to that address. Read more about *making a payment*.

1.1 Creating a wallet

From ElectrumSV 1.3 and beyond, a wallet is now a container for your accounts. This guide shows you how to create an empty wallet with no accounts. After creating the wallet, you will of course want to add an account to it, in order to be able to start using it.

1.1.1 Choosing the location and file name

The first step is to choose where to store your wallet, and what it's file name should be. If you choose not to store your wallet in the default location that ElectrumSV uses, it is likely that you will quickly be able to find it again in the "Recently Opened Wallets" list when you open it again in the future.

Start off at the wallet selection page.

You will be presented with a file dialog that lets you choose where your wallet will be stored, and what it will be named. It defaults to the standard ElectrumSV wallet location on your operating system. Enter a file name, and click "Save" (or press the enter key).

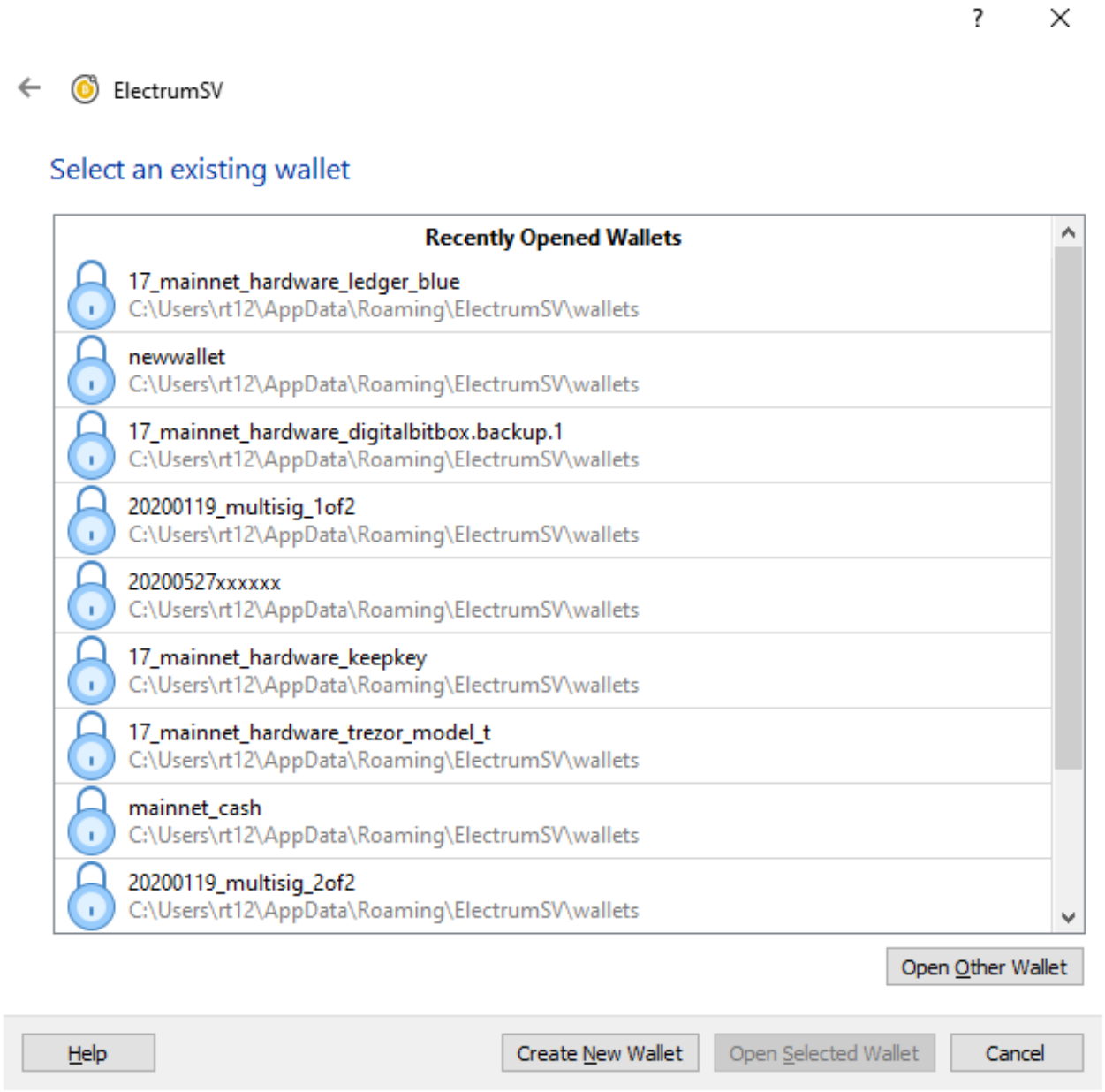

Fig. 1: The wallet selection page.

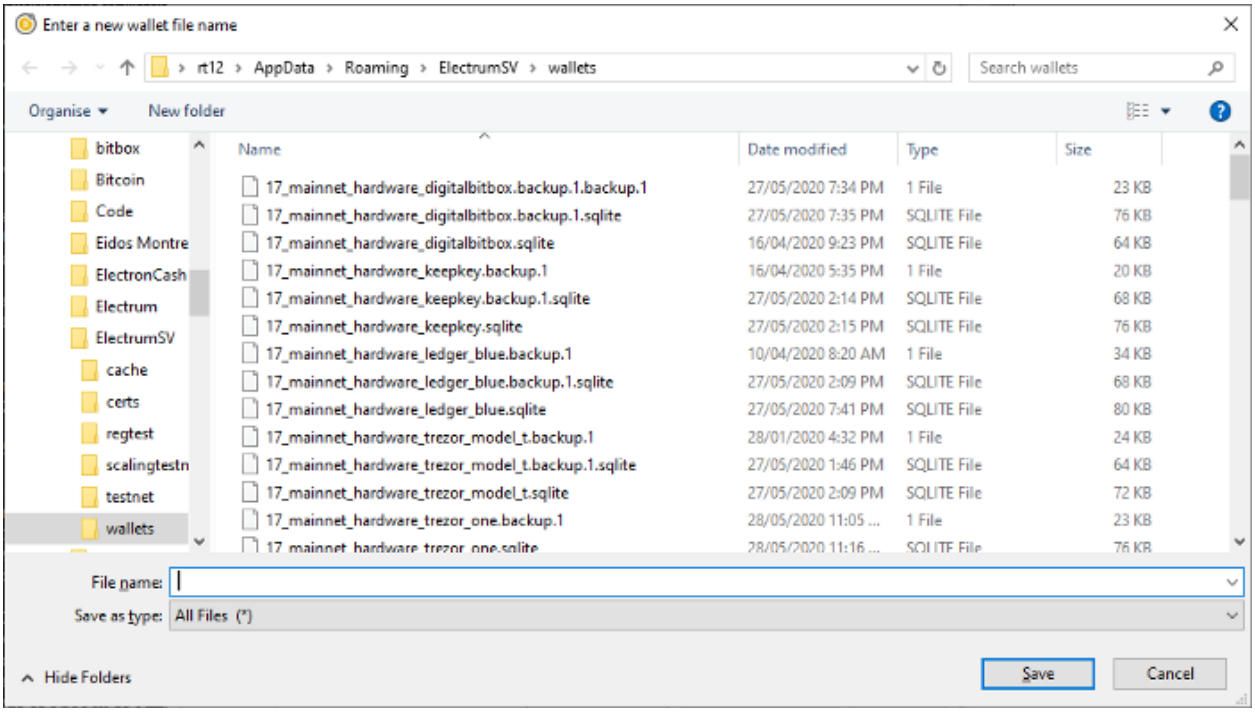

Fig. 2: The wallet file name dialog.

1.1.2 Add a mandatory password

The next step is setting a password for your new wallet. We require a password and there is no way to opt out, but you can always enter something like “password” or “123456” if you wish. This is also required for hardware and watch-only wallets, where there is no key or seed word data to encrypt.

Once you have entered a password, and confirmed it, the “OK” button will become enabled and you can click it (or just press the enter key) to open the new wallet.

Congratulations, you have created a new empty wallet. It will not be usable until you have created an account, and various parts of the user interface will indicate this.

1.2 Creating an account

If you are reading this, you likely have a new wallet that has no accounts, and you want to add one to it. We support addition of a wide variety of account types:

- A new “Standard” account. This is the equivalent of creating a new ElectrumSV seed-word based wallet in 1.2.5 and earlier.
- A multi-signature account. Use this if you are creating a new multi-signature account, or restoring an existing one from master public keys, seed words and so on.
- Importing from text. Use this to import your seed words, whether Electrum seed words, BIP39 seed words from another wallet, private keys, public keys, master public keys, master private keys, and so on.
- Importing a hardware wallet. If you have an existing hardware wallet that has a seed set up on it, then you can use this to add an account that links to it and uses it to sign. If you have a hardware wallet that does not have a seed set up on it, you should also be able to use this to set it up unless the device is a Ledger. Do not buy a Ledger.

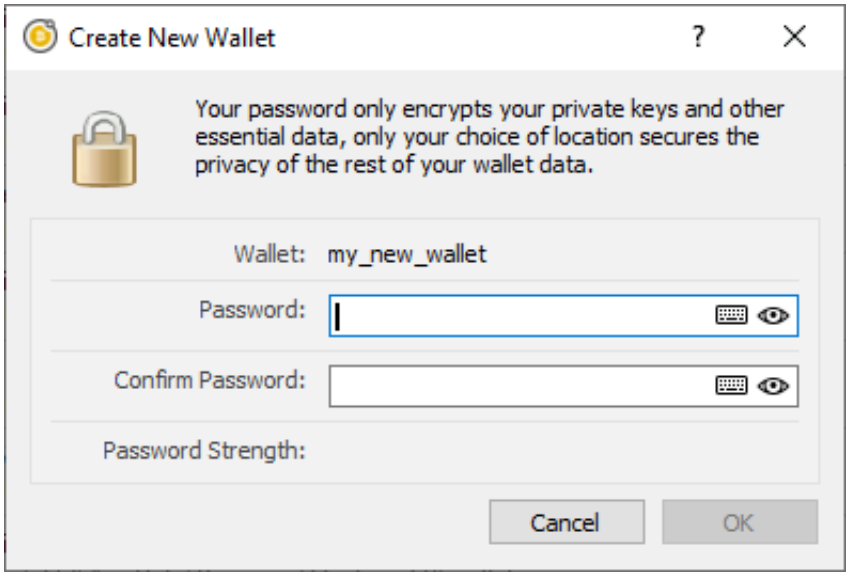

Fig. 3: The wallet file name dialog.

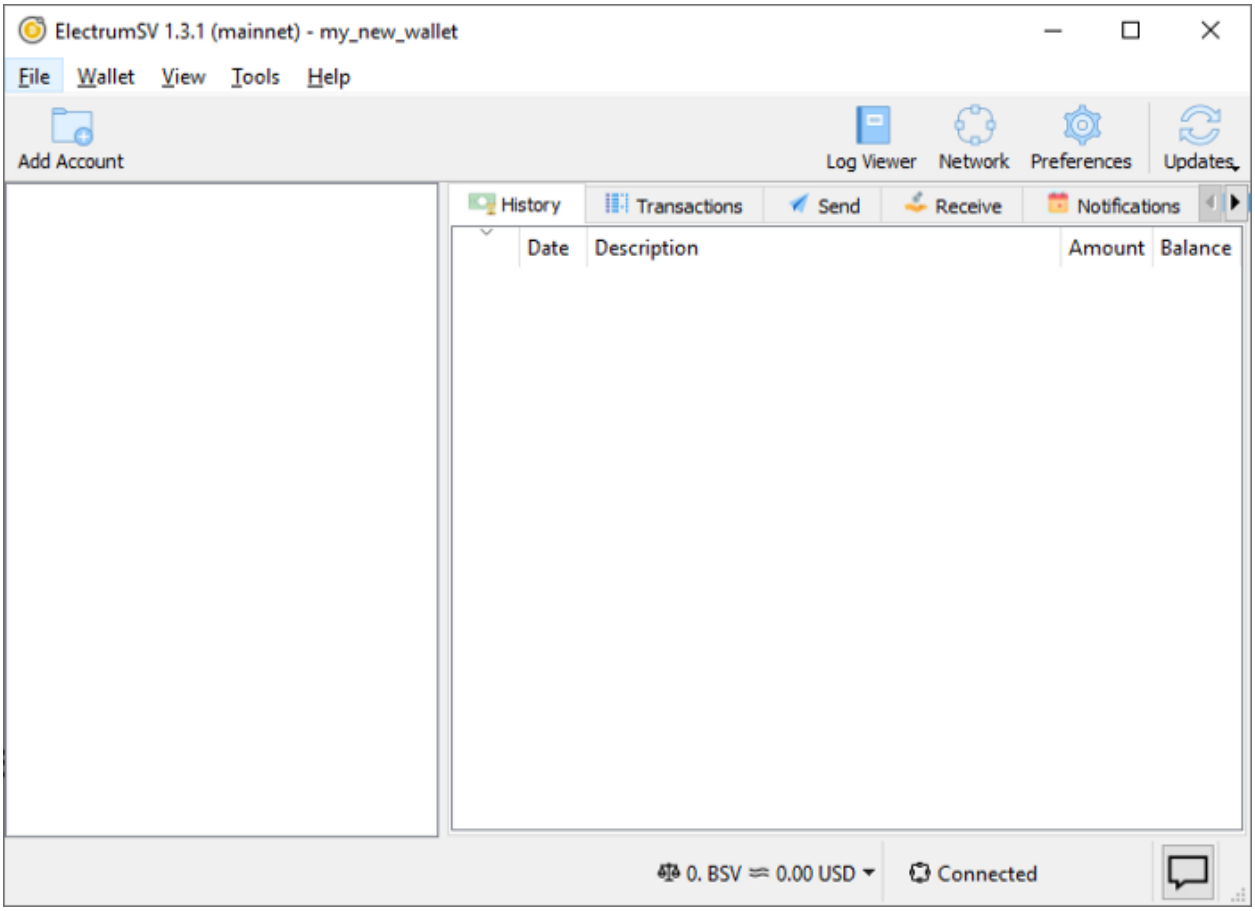

Fig. 4: The new wallet's wallet window.

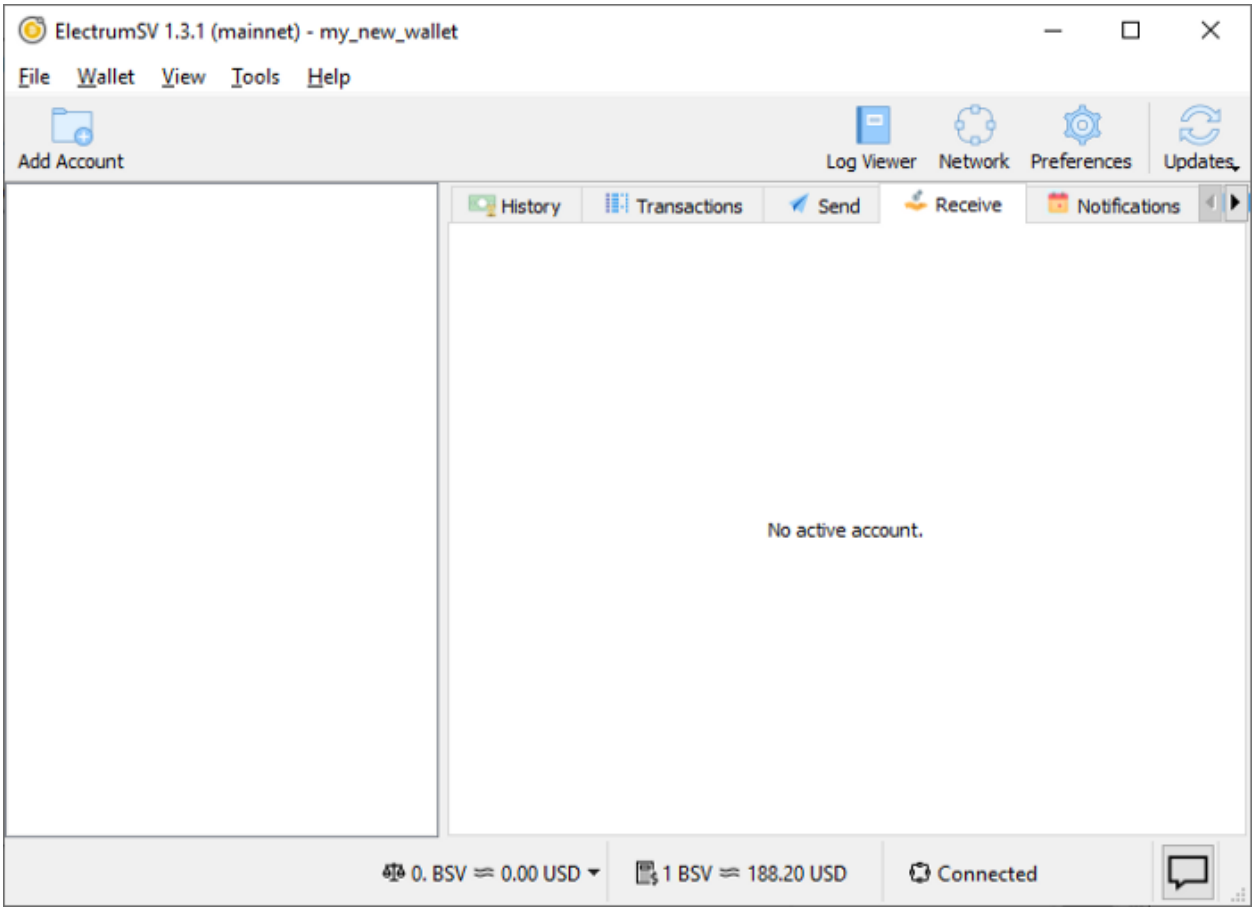

Fig. 5: The receiving tab is disabled.

This guide solely covers creating a “Standard” account.

1.2.1 Adding an account

In the top-left-hand corner of your wallet window, you will see the “Add Account” button. Click it and it will open the account wizard which allows all supported types of accounts to be created.

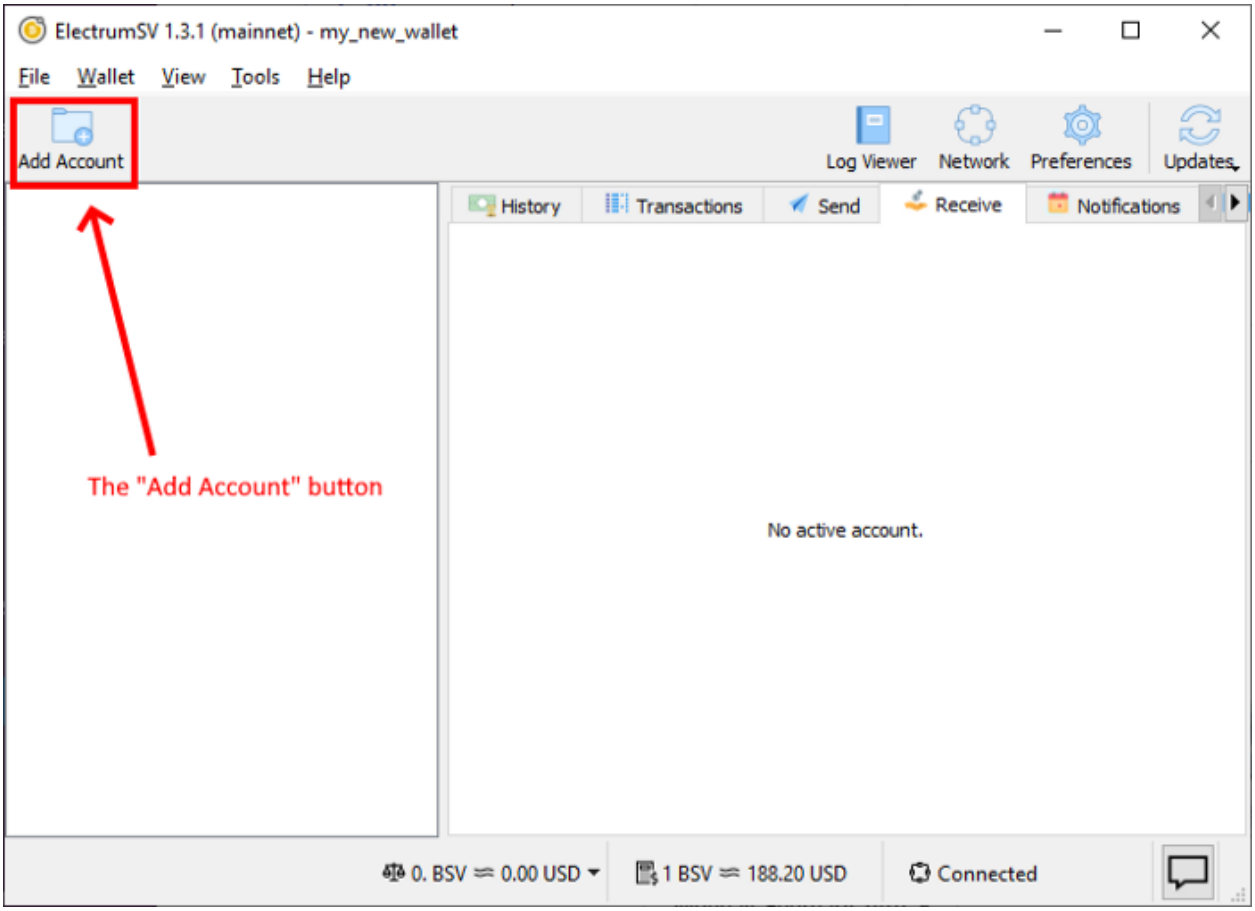

Fig. 6: The “Add Account” button highlighted.

The account wizard offers four different types of account addition, at the time of writing.

1.2.2 Creating a new “Standard” Account

Double-click on the “Standard” entry to proceed. Or if you prefer to work for it, click the “Next” button or press the enter key. You will be asked for your password so that the generated seed words and private key data can be encrypted into your wallet. This also verifies you have the ability to really use this wallet, and should able to add an account.

You will immediately see that the account has been added to your wallet. You will note that at no point did you have to copy down your new seed words, or confirm them. You will be reminded to back them up by the wallet, and can do so at your leisure and own risk.

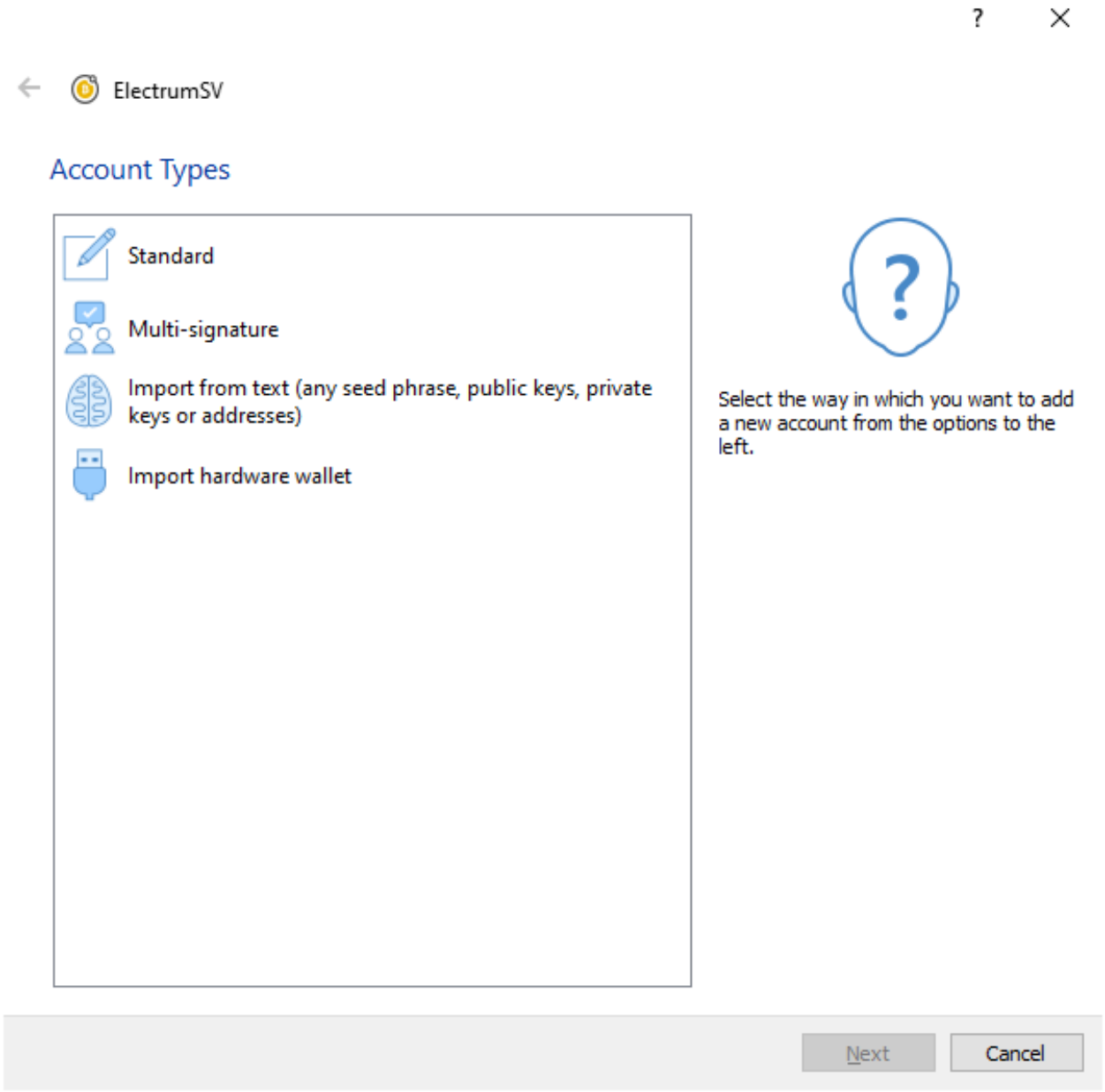

Fig. 7: The Account Types page in the Account Wizard.

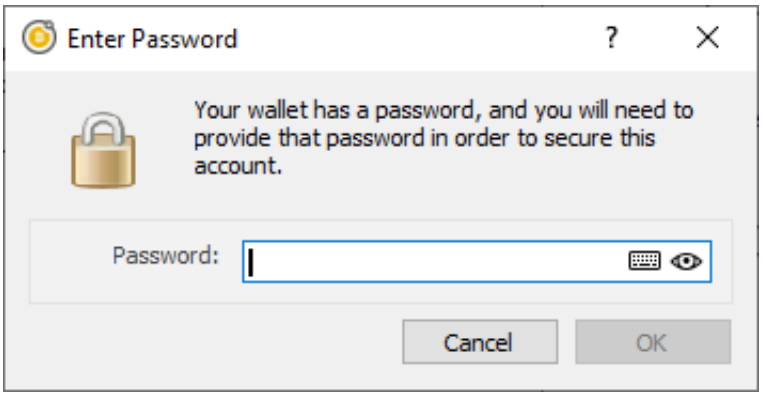

Fig. 8: The password dialog.

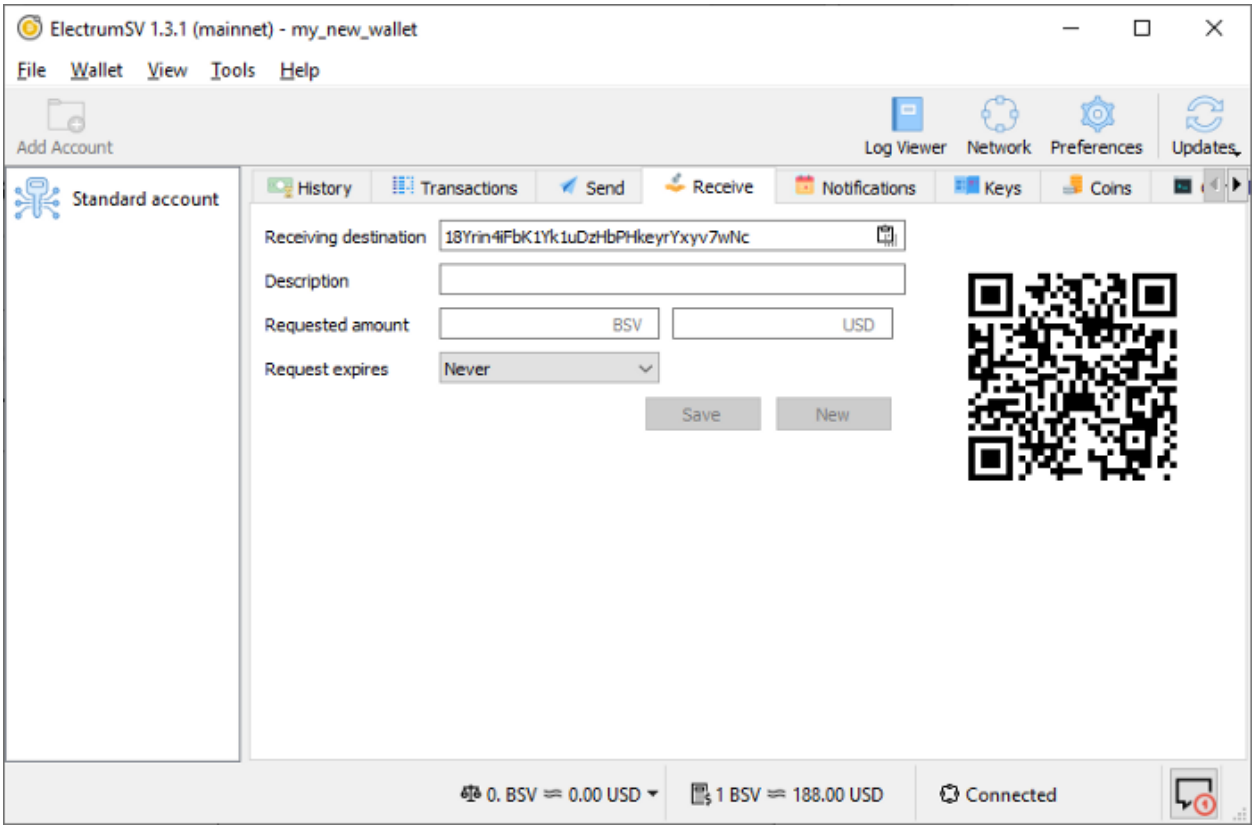

Fig. 9: The new account's receiving tab.

1.2.3 Backing up your seed words

The wallet window now has a notification center, which is used to remind you to deal with important events, and point out how you can do it.

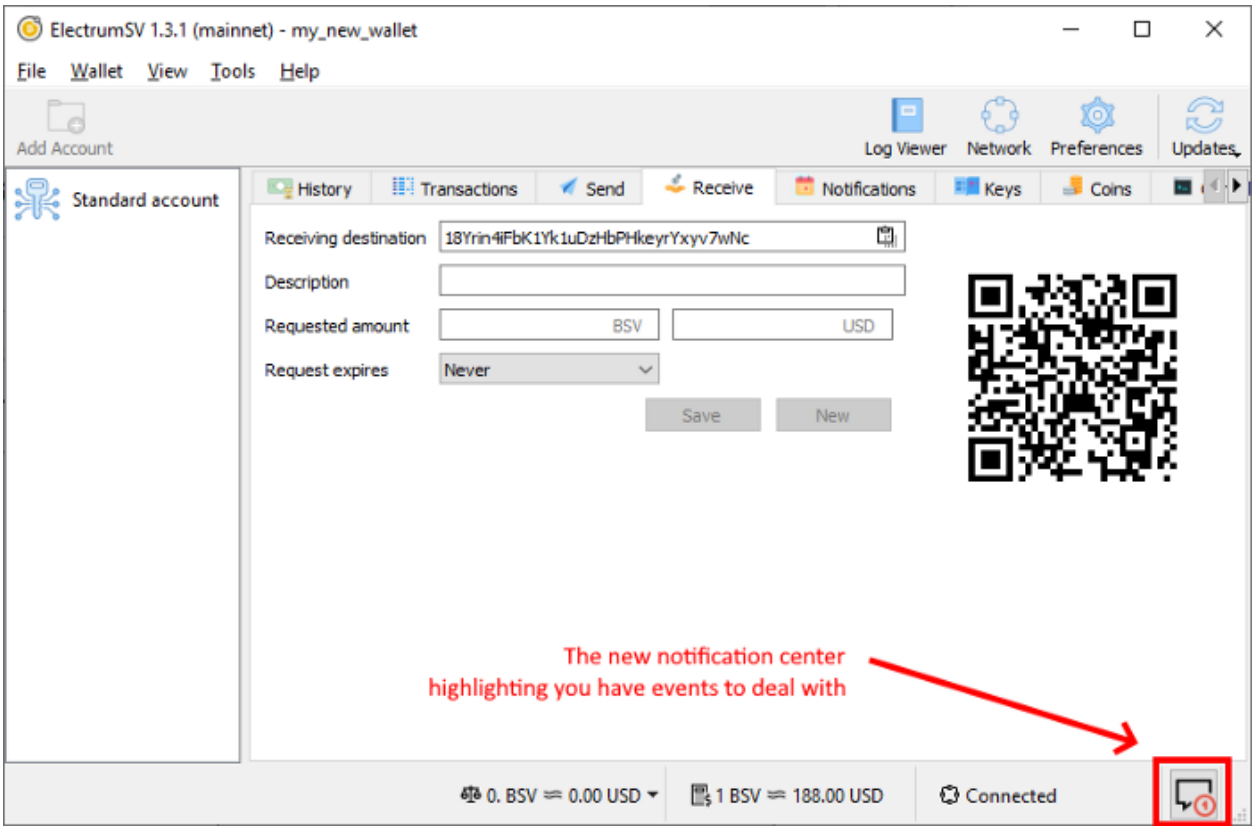

Fig. 10: The wallet's notifications indicator.

1.2.4 The initial backup notification

Clicking the notification icon will make the new "Notifications" tab the active one and show the initial notification about backing up your data.

1.2.5 Follow the link to your secured data

If you click on the "account's secured data" link, it will take you directly to that secured data. But first it will need your password so it can decrypt that data for display.

Having entered the correct password you will see the secured data.

Congratulations, now write down the seed words somewhere safe. I recommend you look into SAFEWORDS to help you with this. You can dismiss the notification by clicking on the "X" in it's top right corner.

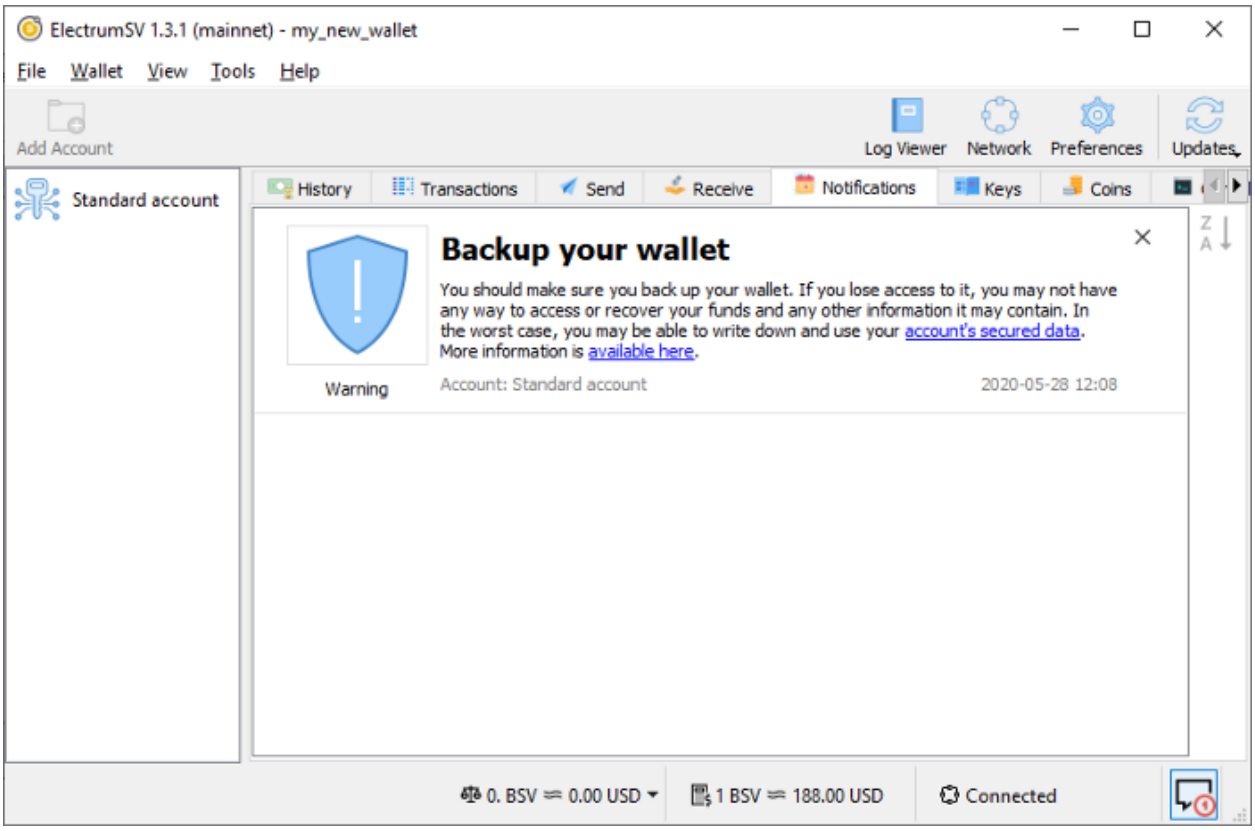

Fig. 11: The initial backup notification in the wallet's notifications tab.

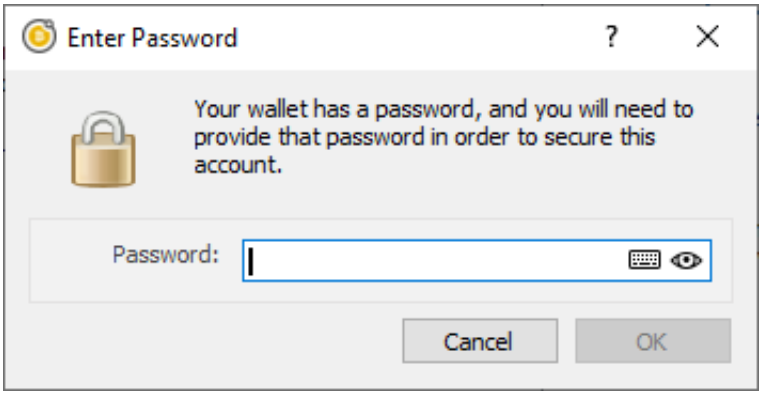

Fig. 12: The password dialog.

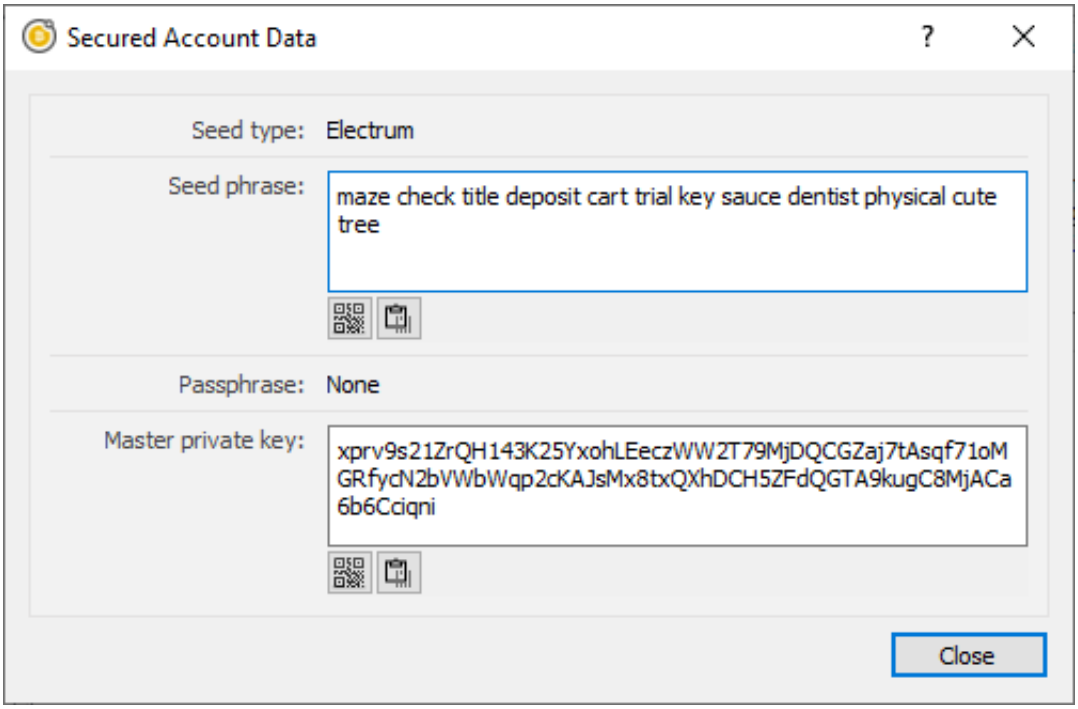

Fig. 13: The secured data dialog.

1.3 Receiving a payment

There are a number of ways in which you can receive payments to a given account in your ElectrumSV wallet. At this time, they all involve having the other party pay to a new address obtained from your account's receive tab.

Two ways that someone can make a payment are:

1. You copy the displayed address and give it out.
2. The other party takes a photo of the displayed QR code and their wallet software lets them pay to it.

1.3.1 Giving out an address

The oldest way to receive a payment is to give out an address from your wallet to the other party, and then wait for them to pay you. In the future, this will not be supported, but for now it is. The shown address is automatically replaced with another address, as the wallet detects that the shown address was used in an incoming payment. This kind of works, but not really, to assist the user in always giving out a new fresh previously unused address.

Important: The flaw in paying to addresses is that the other party has no way to know that the address they get, is the one that you tried to give them. Because they look like random letters and numbers it is possible that they can be replaced without either party knowing before it is too late. While reports of this happening are rare, it might be worth taking precautions to make sure this does not happen to you.

You can copy the address, paste it in an email and send it to the recipient. Paste it into a chat application. Or get it to them in any number of possible ways.

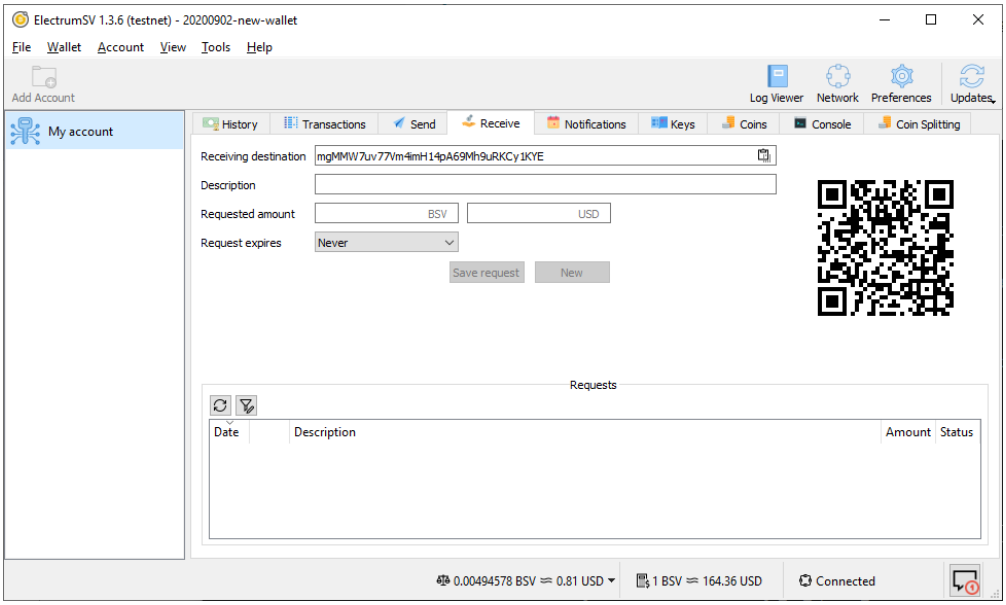

Fig. 14: The receiving tab in a standard account.

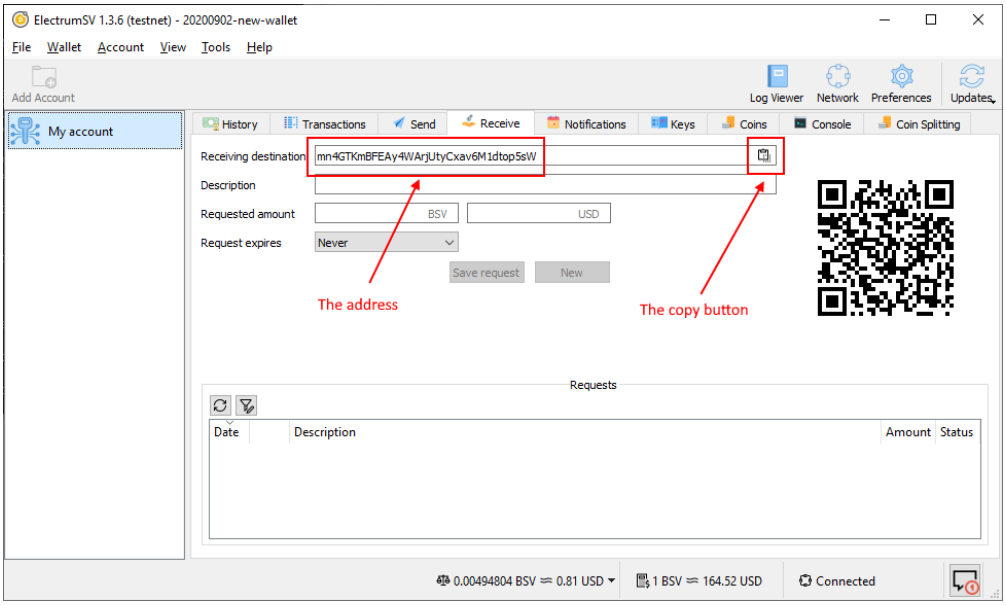

Fig. 15: The new address offered in the receiving tab.

1.3.2 Using a QR code

If the other party is standing there with you, you can show them the receiving tab and they can take a photo of the QR code with their wallet. Their wallet will extract the address and streamline the payment process. You can fill out the fields with a requested amount to also include that in the QR code, which further streamlines the process.

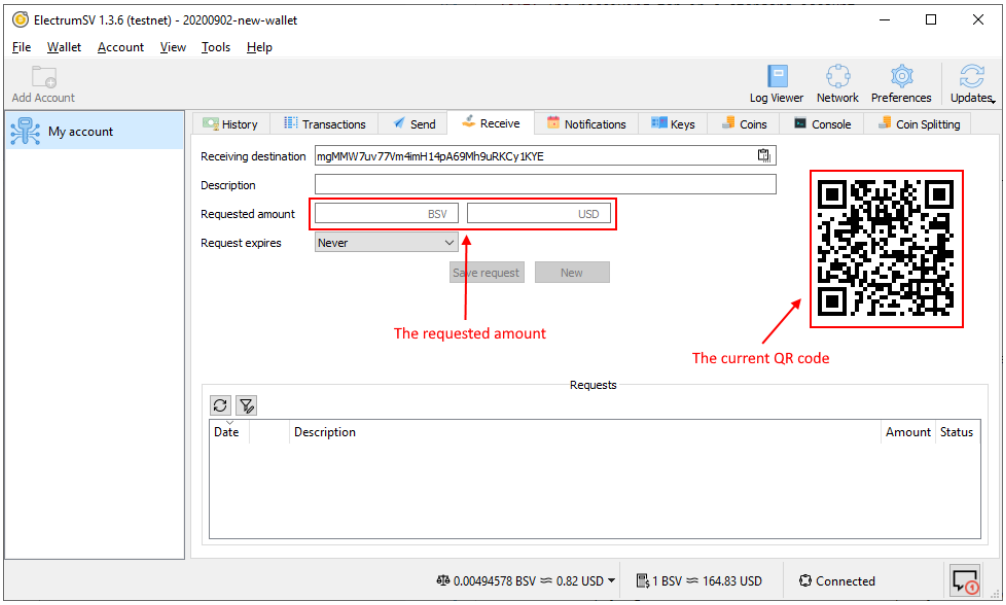

Fig. 16: The QR code provided in the receiving tab.

1.3.3 Identifying incoming payments

In the legacy model, which is still the most common one, payments are fire and forget. The payer constructs a transaction and broadcasts it to the blockchain. Then when your wallet gets a notification a payment of interest has appeared in the blockchain, it retrieves that transaction and factors it into the related account.

With this model, the wallet has no idea a payment is incoming until it arrives out of the blue. A new and better model is available in the form of Paymail, but ElectrumSV does not have the service infrastructure to support it at this time. We are however working towards it.

1.4 Making a payment

If you are reading this, you probably want to know how to make a payment. We currently only support making payments in the following ways:

- Payment to a Bitcoin address.
- Payment to a BIP276 address.
- Payment to a Bitcoin script.

This guide solely covers payment to an address. It is not recommended you pay to a Bitcoin script unless you are an expert.

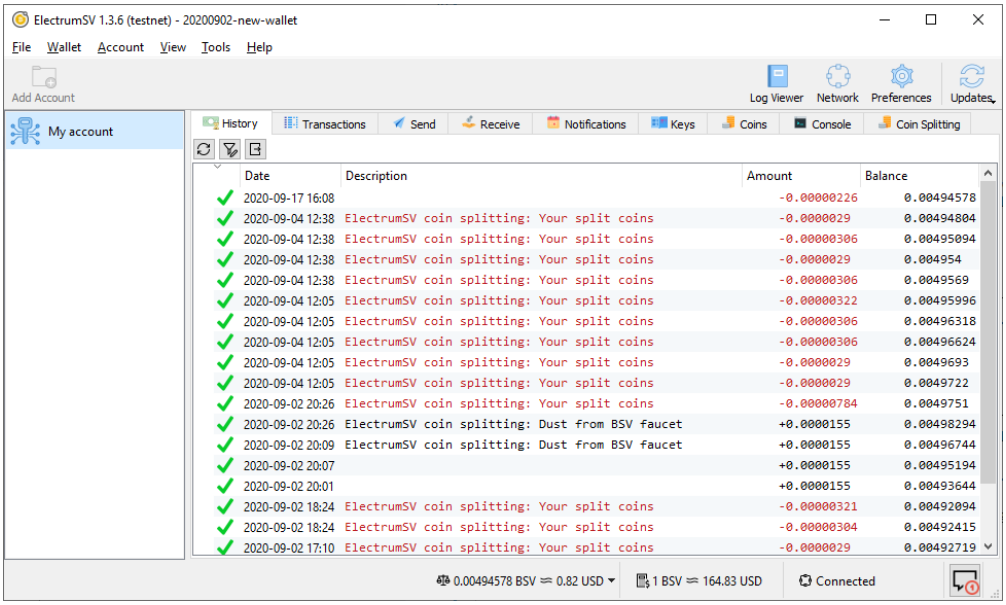

Fig. 17: The history tab when awaiting an incoming payment.

1.4.1 Paying to yourself

At this point you should have a wallet with a standard account. You should also have an address from another party, that you can make a payment to. However for the purpose of this guide, you can make a payment to yourself, if you have no-one else to currently pay.

Start off on the receiving tab.

As you will be paying to yourself, copy the shown address. The best way to do this is to click on the copy button, which will copy it to the clipboard. You will use this address as you would the address for any other party.

1.4.2 Paying to an address

Ensure your wallet window is now showing the send tab. Select the "Pay to" field and paste in the address you wish to make a payment to.

After pasting in the address, enter a nominal amount of Bitcoin SV to send, where your wallet has sufficient funds to do so.

Important: If you are paying to addresses a good practice is to make what is called a `pilot payment` first, where you pay a small amount you can afford to lose, before paying the larger full amount.

Click the "Send" button to start the payment process.

Ensure that both the amount you are sending and the mining fee are the appropriate amounts, then enter your password and click "OK". The "OK" button only becomes enabled when you have entered your password correctly. The transaction will broadcast, and you should receive a confirmation that the payment was made.

The confusing sequence of letters and numbers is actually the ID of the transaction that contained your payment. This can be used to look up your payment, if you were to take it and paste it into a web site that indexes the Bitcoin SV blockchain.

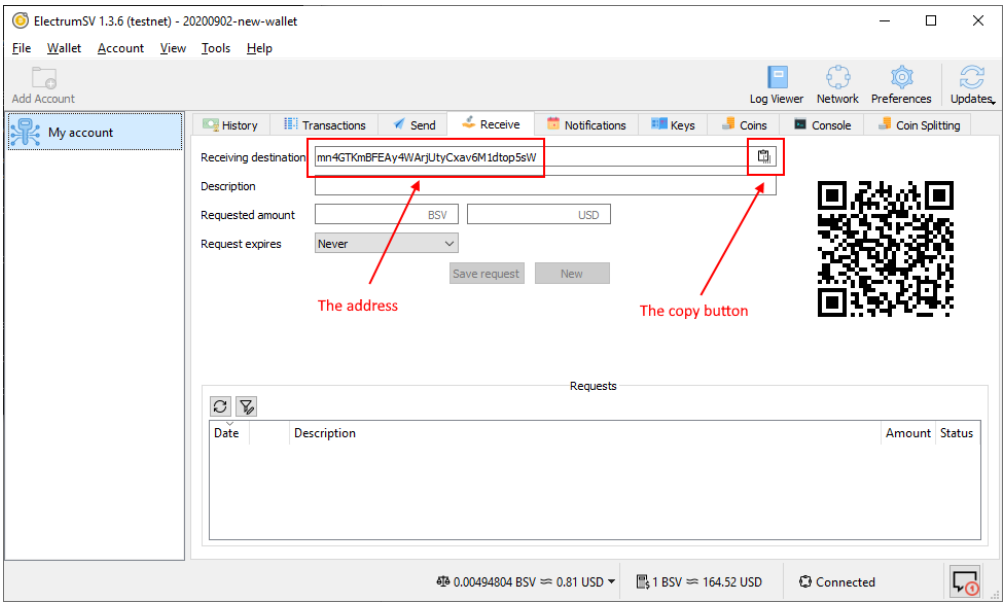

Fig. 18: Highlighted areas on the receiving tab.

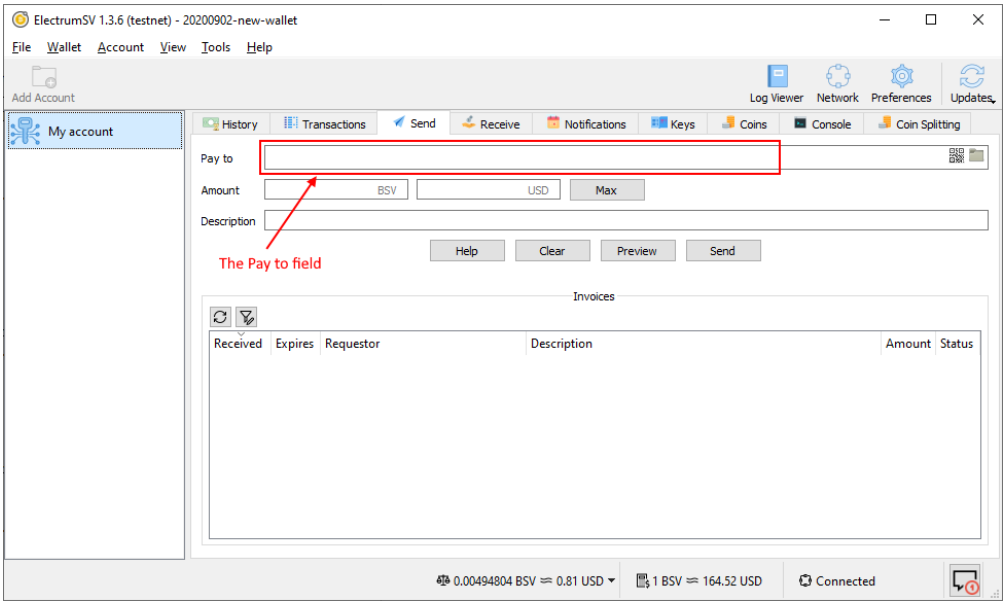

Fig. 19: Highlighted areas on the send tab.

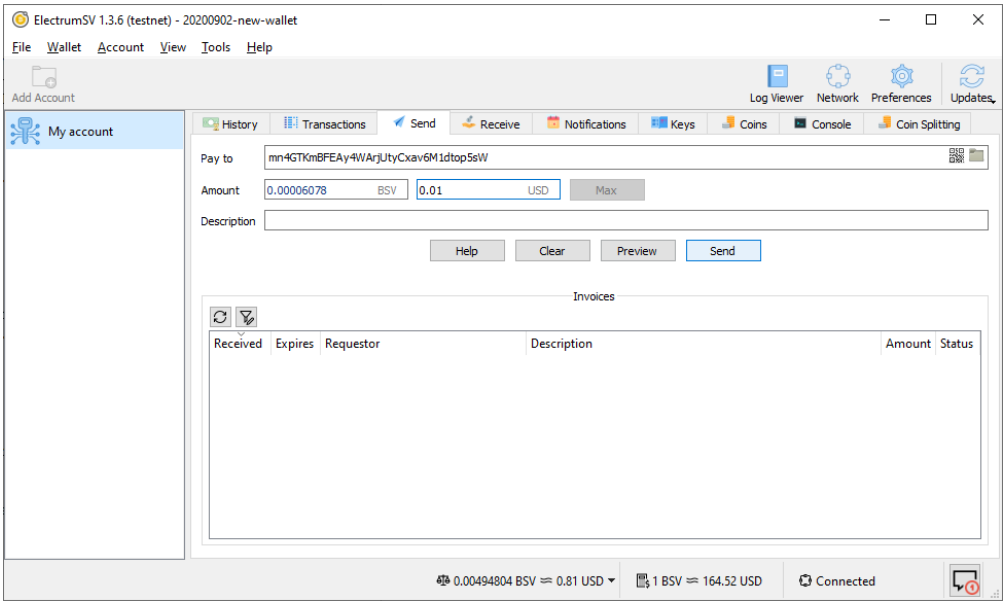

Fig. 20: The filled out send tab.

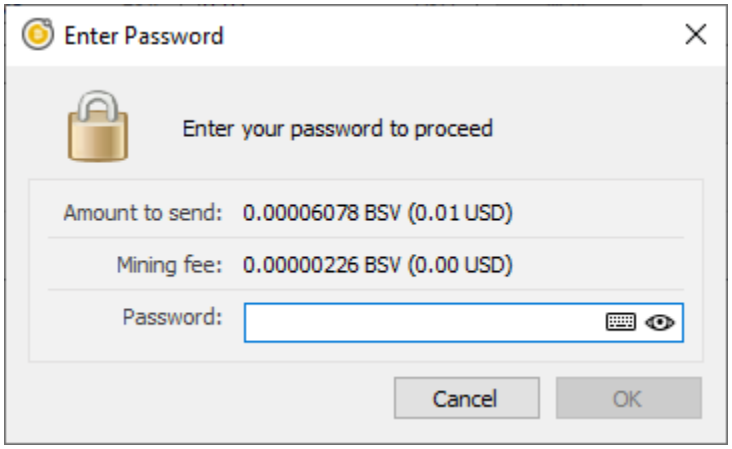

Fig. 21: The password confirmation dialog.

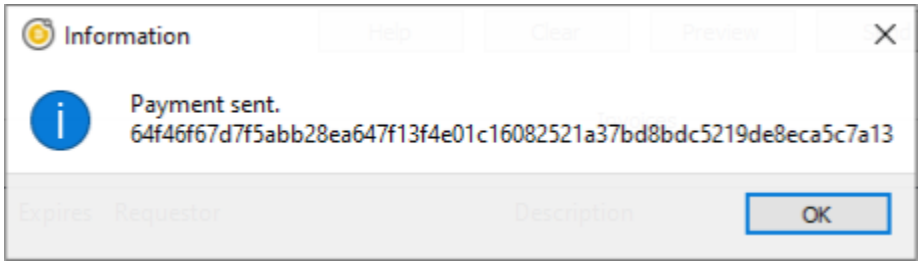

Fig. 22: The payment sent dialog.

1.4.3 The record of payment

At it's current state of development, the wallet does not have much context about payments made. But you can see the transactions this account is involved in, in the history tab.

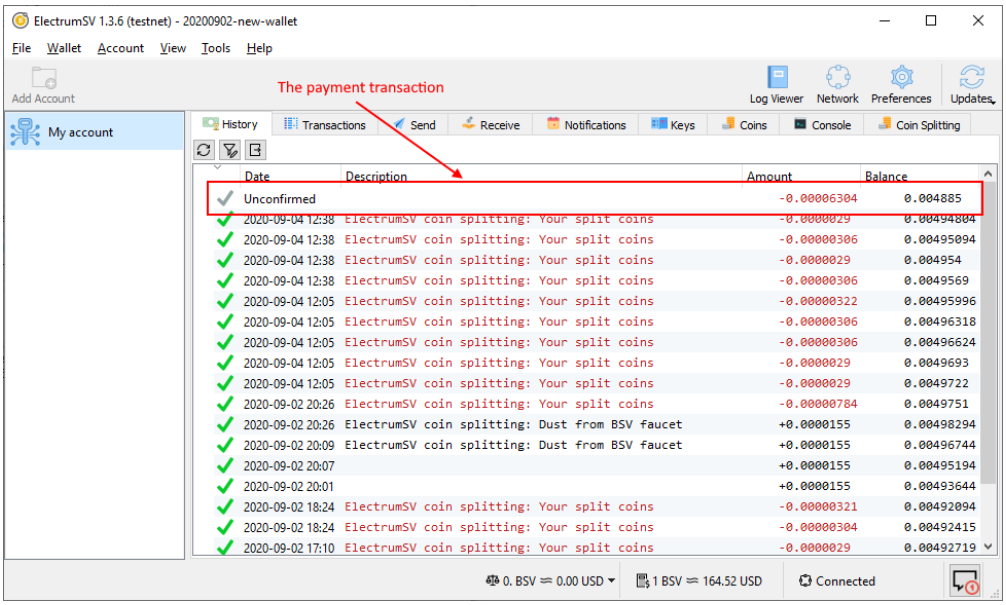

Fig. 23: The highlighted payment transaction in the history tab.

If you had provided a description when making the payment, it would appear here in much the same way as the existing transactions with their "ElectrumSV coin splitting: Your split coins" descriptions.

CHAPTER

TWO

PROBLEM SOLVING

Why doesn't my hardware wallet work? Hardware wallet makers do not provide anywhere near enough support for their devices, and some have a history of making breaking changes that stop them working in ElectrumSV. If your hardware wallet does not work then this is where you should look for some pointers, whether the device is a Trezor, a Ledger, a Keepkey or a Bitbox. Read more about *hardware wallet issues*.

2.1 Hardware wallet issues

2.1.1 Ledger

Message: "The sign path is unusual"

Ledger have implemented enforced derivation paths for each cryptocurrency. If you set up your hardware wallet with a different derivation path than it expects for the app you are using on your Ledger device, then it will in theory show you the above message warning you. You can read the reasons why on the article linked above. However, some of our users have reported that this is not a warning and in fact it prevents them from signing leaving them unable to access their funds.

It is recommended that users who experience this upgrade their Ledger firmware to the latest version and if it still does not allow them to sign, then work out some way to get the funds from their Ledger back into it with the derivation path Ledger expects. It is very likely that if users were unable to sign, given that Ledger say it should only be a warning, that this was a temporary bug in their firmware and an upgrade should fix it when they do.

The process of moving funds to the correct derivation path might be done as follows:

1. Make a new wallet and account in ElectrumSV using the text account option. This will involve entering your seed words from your Ledger, so that you can manage the coins directly in ElectrumSV.
2. Verify that you can see your coins in your new account, and send a small amount back to yourself to ensure you have access.
3. Make a second wallet and account in ElectrumSV using the hardware wallet option. Ensure you use the derivation path that Ledger expects you to use, whatever that is.
4. Send a small amount from the first (imported text words) account to the second (new hardware wallet) account. Verify that it arrives. This is intended to put an existing small amount of coins in your new hardware wallet account so you can verify it works correctly.
5. Send a small amount from the new hardware wallet account back to itself. Verify that the hardware wallet signs the transaction correctly as it has in the past, and the problem is solved.
6. Send all the coins remaining in the imported text words account over to the new hardware wallet account. You should now have your funds safely stored in your Ledger again.

This process of moving your coins of course completely bypasses the protection that your hardware wallet was supposed to provide, but there's not much else you can do if you want to continue using it and it won't otherwise let you.

2.1.2 Trezor

Message: "Signing transaction" never goes away

In order to address flaws in the Bitcoin Core protocol, Trezor made changes to transaction signing which causes errors when users try to sign transactions in ElectrumSV. This means that anyone who updates their Trezor firmware will not be able to sign transactions.

Let us be clear. It appears that Trezor decided to break all transaction signing for everyone in order to fix a Bitcoin Core problem. They could have chosen to have Bitcoin Core users opt-in to the fixes, and not casually broken all other transaction signing for all their users who use other cryptocurrencies.

The changes to fix this are comprehensive. This is part of a recent pattern of hardware wallet makers making changes that break their customers ability to use their devices. It is very likely we will fix this, but we have a lot of other more pressing work to do and the cost of getting side-tracked for this is too high at this time.

Trezor Model T

Supported firmware version trezor-2.3.0.bin

Firmware release data Trezor Model T feed

If you are using a later firmware than the recommended version, you will need to downgrade. Make sure that you have backed up your seed words, as this may cause the device to be wiped.

Trezor One

Supported firmware version trezor-1.9.0.bin

Firmware release data Trezor One feed

If you are using a later firmware than the recommended version, you will need to downgrade. Make sure that you have backed up your seed words, as this may cause the device to be wiped.

CHAPTER

THREE

BUILDING ON ELECTRUMSV

How can I access my wallet using the REST API? For most users, accessing their wallet with the user interface will be fine. But if you have a minimal amount of development skill the availability of the REST API gives you a lot more flexibility. The REST API allows a variety of actions among them loading multiple wallets, accessing different accounts, obtaining payment destinations or scripts from any of the accounts. Perhaps you want to add your own interface for your wallet or maybe automate how you use it. Read more about the *REST API*.

How would I extend ElectrumSV as a customised wallet server? The REST API is limited in what it can do by nature. Getting the ElectrumSV development team to add what you want to it, is not guaranteed to happen, may not even be possible and if it was who knows how long it would take. An alternative is to build your own "daemon application" which is a way of extending ElectrumSV from the inside. Read more about *customised wallet servers*.

Do I have to develop against the existing public blockchains? ElectrumSV provides a way for developers to do offline or local development. *customised wallet servers*.

3.1 The REST API

Technically, the restapi is an example 'dapp' (daemon application). But is nevertheless provided in a format that aims to eventually cover the majority of basic use cases.

This RESTAPI may be subject to slight changes but the example dapp source code is there for users to modify to suit your own specific needs.

3.1.1 Endpoints

get_all_wallets

Get a list of all available wallets

Method GET

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets`

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets`

get_parent_wallet

Get a high-level information about the parent wallet and accounts (within the parent wallet).

Method GET

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets/{wallet_name}`

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets/worker1.sqlite`

load_wallet

Load the wallet on the daemon (i.e. subscribe to ElectrumX for active keys) and initiate synchronization. Returns a high-level information about the parent wallet and accounts.

Method POST

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets/{wallet_name}`

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets/worker1.sqlite`

get_account

Get high-level information about a given account

Method POST

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets/{wallet_name}/{account_id}`

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets/worker1.sqlite/1`

get_coin_state

Get the count of cleared, settled and matured coins.

Method GET

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets/{wallet_name}/{account_id}/utxos/coin_state`

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets/worker1.sqlite/1/utxos/coin_state`

get_utxos

Get a list of all utxos.

Method GET

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets/{wallet_name}/{account_id}/utxos`

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets/worker1.sqlite/1/utxos`

get_balance

Get account balance (confirmed, unconfirmed, unmatured) in satoshis.

Method GET

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets/{wallet_name}/{account_id}/balance`

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets/worker1.sqlite/1/utxos/balance`

remove_txs

Removes transactions in the 'Signed' state.

Deleting transactions in the 'Dispatched', 'Cleared', 'Settled' states could cause issues and so is not supported at this time (a DisabledFeatureError will be returned). If you require this feature, please make contact via the Atlantis Slack or the MetanetICU slack.

Method POST

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets/{wallet_name}/{account_id}/txs/remove`

Request Body Payload

```
{
    "txids": [<txid1>, <txid2>, ...]    (optional field)
}
```

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets/worker1.sqlite/1/txs/remove`

Sample Body Payload

```
{
    "txids": ["d45145f0c2ff87f6cfe5524d46d5ba14932363e927bd5a4af899a9b8fc0ab76f"]
}
```

Sample Response

```
{
    "value": {
        "message": "All StateSigned transactions in set: [
↪'299405452db66866b9fed2ebe83bee5d41c4a29a0d88e2f8590f1ced7f5531b1'] deleted␣
↪fromTxCache, TxInputs and TxOutputs cache and SqliteDatabase."
    }
}
```


get_transaction_history

Get transaction history.

Method POST

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets/{wallet_name}/{account_id}/txs/history`

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets/worker1.sqlite/1/txs/history`

Sample Response


```
{
    "value": [
        {
            "txid": "d45145f0c2ff87f6cfe5524d46d5ba14932363e927bd5a4af899a9b8fc0ab76f
↪",
            "height": 201,
            "timestamp": "2020-09-30T21:02:32",
            "value": "+25.",
            "balance": "25.",
            "label": "",
            "fiat_value": "No data",
            "fiat_balance": "No data"
        }
    ]
}
```


get_transactions_metadata

Get transaction metadata.

Method POST

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets/{wallet_name}/{account_id}/txs/metadata`

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets/worker1.sqlite/1/txs/metadata`

Sample Request Payload

```
{
    "txids": ["d45145f0c2ff87f6cfe5524d46d5ba14932363e927bd5a4af899a9b8fc0ab76f"]
}
```

Sample Response

```
{
    "value": {
        "d45145f0c2ff87f6cfe5524d46d5ba14932363e927bd5a4af899a9b8fc0ab76f": {
            "block_id":
↪"7a24a95c4bfec88785203dc2e36dcf4493469d4d8cadfd4e89b37f7eae9e77bd",
            "height": 201,
            "conf": 1,
            "timestamp": 1601452952
        }
    }
}
```

fetch_transaction

Get the raw transaction for a given hex txid (as a hex string) - must be a transaction in the wallet's history.

Method POST

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets/{wallet_name}/{account_id}/txs/fetch`

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets/worker1.sqlite/1/txs/fetch`

Sample Request Payload

```
{
    "txid": "d45145f0c2ff87f6cfe5524d46d5ba14932363e927bd5a4af899a9b8fc0ab76f"
}
```

Sample Response

```
{
    "value": {
        "tx_hex":
↪"0200000001adc7943687d0f89c1e20bb1c196e16cd5f08449e5aa7e744c83cc5f67ffe1e6d000000006a47304402204a23
↪"
    }
}
```

create_tx

Create a locally signed transaction ready for broadcast. A side effect of this is that the utxos associated with the transaction are allocated for use and so cannot be used in any other transaction.

Method POST

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets/{wallet_name}/{account_id}/txs/create`

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets/worker1.sqlite/1/txs/create`

Sample Request Payload This example is of a single "OP_FALSE OP_RETURN" output with "Hello" encoded in Hex ("48656c6c6f") the preceeding 0x05 byte represents a pushdata op code to push the next 5 bytes onto the stack (in this case "48656c6c6f").

Additional outputs for leftover change will be created automatically.

```
{
    "outputs": [
        {"script_pubkey":"006a0548656c6c6f", "value": 0}
    ],
    "password": "test"
}
```

Sample Response

```
{
    "value": {
        "tx_hex":
↪"0200000001adc7943687d0f89c1e20bb1c196e16cd5f08449e5aa7e744c83cc5f67ffe1e6d000000006a47304402204a2
↪"
    }
}
```

broadcast

Broadcast a rawtx (created with the previous endpoint).

Method POST

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets/{wallet_name}/{account_id}/txs/broadcast`

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets/worker1.sqlite/1/txs/broadcast`

Sample Request Payload This example is of a single "OP_FALSE OP_RETURN" output with "Hello" encoded in Hex ("48656c6c6f") the preceeding 0x05 byte represents a pushdata op code to push the next 5 bytes onto the stack (in this case "48656c6c6f").

Additional outputs for leftover change will be created automatically.

```
{
    "rawtx":
↪"0100000001b131557fed1c0f59f8e2880d9aa2c4415dee3be8ebd2feb96668b62d45059429010000006b4830450221008
↪"
}
```

Sample Response

```
{
    "value": {
        "txid": "53b1b2886f038183199f3dc6979c9c54934ebe74166e20addb0f318165d1b7ce"
    }
}
```

create_and_broadcast

Atomically creates and broadcasts a transaction. If any errors occur, the intermediate step of creating a signed transaction will be reversed (i.e. the transaction will be deleted and the utxos freed for use).

Method POST

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets/{wallet_name}/{account_id}/txs/create_and_broadcast`

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets/worker1.sqlite/1/txs/create_and_broadcast`

Sample Request Payload This example is of a single "OP_FALSE OP_RETURN" output with "Hello" encoded in Hex ("48656c6c6f") the preceeding 0x05 byte represents a pushdata op code to push the next 5 bytes onto the stack (in this case "48656c6c6f").

Additional outputs for leftover change will be created automatically.

```
{
    "outputs": [
        {"script_pubkey":"006a0548656c6c6f", "value": 0}
    ],
    "password": "test"
}
```

Sample Response

```
{
    "value": {
        "txid": "7a77e888bb9a60f277cf3ae570c1fb61f99c13c9335170895efa07c6a923c91c"
    }
}
```

split_utxos

Creates and broadcasts a coin-splitting transaction i.e. it breaks up existing utxos into a specified number of new utxos with the desired "split_value" (satoshis). "split_count" represents the maximum number of splitting outputs for the transaction. "desired_utxo_count" determines when the desired utxo count has been reached (i.e. if you have 200 utxos but "desired_utxo_count" is 220 then the next coin splitting transaction will create 20 more utxos.

Method POST

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets/{wallet_name}/{account_id}/txs/split_utxos`

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets/worker1.sqlite/1/txs/split_utxos`

Sample Request Payload

```
{
    "split_value": 10000,
    "split_count": 100,
    "password": "test",
    "desired_utxo_count": 1000
}
```

Sample Response

```
{
    "value": {
        "txid": "7a77e888bb9a60f277cf3ae570c1fb61f99c13c9335170895efa07c6a923c91c"
    }
}
```

3.1.2 Regtest only endpoints

If you try to access these endpoints when not in RegTest mode you will get back a 404 error because the endpoint will not be available.

topup_account

Tops up the RegTest wallet from the RegTest node wallet (new blocks may be generated to facilitate this process).

Method POST

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets/{wallet_name}/{account_id}/topup_account`

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets/worker1.sqlite/1/topup_account`

Sample Request Payload

```
{
    "amount": 10
}
```

Sample Response

```
{
    "value": {
        "txid": "cea035abf5b8c6814db2b3ab4240a7c8f65ea08d8b3a32a0bdb1d6c0605bb7e0"
    }
}
```

generate_blocks

Tops up the RegTest wallet from the RegTest node wallet (new blocks may be generated to facilitate this process).

Method POST

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets/{wallet_name}/{account_id}/generate_blocks`

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets/worker1.sqlite/1/generate_blocks`

Sample Request Payload

```
{
    "nblocks": 3
}
```

Sample Response

```
{
    "value": {
        "txid": [
            "410a6fd9024613d8e98953706b31f13ed875a7dfd9f2cee39b33ed2de0a15c92",
            "262b113c711eb11e8a44b58aea8be36ba788b599a2089b425d0eb7f94d7d3913",
            "12a972760942e24b53d74c18608a16aeef6df3d193a80e5f503d1457b1fb815a"
        ]
    }
}
```

create_new_wallet

This will create a new wallet - in this example "worker1.sqlite". This example was produced via the electrumsv-sdk which allows a convienient method for running a RegTest node, electrumX instance (pre-configured to connect) and an ElectrumSV instance with data-dir=G:\electrumsv_official\electrumsv1.

Method POST

Content-Type application/json

Endpoint `http://127.0.0.1:9999/v1/{network}/dapp/wallets/{wallet_name}/{account_id}/create_new_wallet`

Regtest example `http://127.0.0.1:9999/v1/regtest/dapp/wallets/worker1.sqlite/create_new_wallet`

Sample Request Payload

```
{
    "password": "test"
}
```

Sample Response

```
{
    "value": {
        "new_wallet": "G:\\electrumsv_official\\electrumsv1\\regtest\\wallets\\
↪worker1.sqlite"
    }
}
```

3.2 Wallet as a service

As the Bitcoin SV ecosystem matures services will become available that allow businesses to outsource the wallet management and the services necessary for their products. There is a sizeable advantage to this, as it allows the business to focus on the products and avoid complicated and rather standard work that there is a benefit to outsourcing.

For the businesses that want or even need to be in control of their own wallet and infrastructure, they can treat ElectrumSV as a common open source base, and extend it with their own proprietary functionality. An starting point for this approach can be found in the example application on Github.

This documentation will be fleshed out as time allows.

3.3 Local or offline development

An command-line based environment is provided for local Bitcoin SV development. There is no requirement that any developer using it needs to be online while they use it, which is fully compatible with the very flexible ability to do offline development.

More details will be provided as polishing is completed.

CHAPTER

FOUR

THE ELECTRUMSV PROJECT

Perhaps you are a developer who already helps out on the ElectrumSV project, or you who would like to get involved in some way, or you are just curious about the processes and information related to project management and development. If so, this is the information you want.

How can you contribute? There are many ways that you can help the ElectrumSV project improve. If you want something to work in a different way, you can work on making it different and offer us the changes. If you feel the documentation could be better, you can improve it and offer us the changes. If you want ElectrumSV or anything related to it in your native language, you can offer to do the work to translate it. And that's just a few of the possibilities. Read more about *contributing*.

Where is the continuous integration and how is it used? We use Microsoft's Azure DevOps services for continuous integration. Microsoft provide generous levels of free usage to open source projects hosted on Github. This is used to do a range of activities for every change we make to the source code, from running the unit tests against each change on each supported operating system, to creating a packaged release for each system that can be manually tested. Read more about our use of *continuous integration*.

What is the process of releasing a new version? Because we generate packaged releases for every change we make, with a bit of extra work we can generate properly prepared public releases. This involves changing the source code so that the release has the content changes required for new version, and also publishing the release and updating the web site to have the content changes required to offer it for download. Read more about the *release process*.

4.1 How you can contribute

What are some of the ways you might contribute to ElectrumSV?

- Contributing translations.
- Contributing new features.
- Contributing bug fixes.
- Reporting problems.

4.1.1 Translations

Anyone wishing to contribute translations of the text in the ElectrumSV user interface, can do so by the ElectrumSV project on Crowdin. Once you've done entering translations let us know, and we'll do the process of exporting the latest data from Crowdin so that your transaction work gets used.

4.1.2 New features

Be aware that you should check with us before starting work on a feature you are hoping we will accept into ElectrumSV. If we accept a new feature, we then have to maintain it and accept the extra work involved on top of that required for support requests, current features and bug fixes. And it may be that depending on the feature we cannot remove it later, if users become reliant on it or have data that requires it to be present.

4.1.3 Bug fixes

We welcome bug fixes for existing problems, whether they are problems you encounter yourself or ones that you see others have reported that have not already been fixed. You can find our existing bugs in our issue tracker on Github.

4.1.4 Reporting problems

Even if you do not have the experience, skill or inclination to attempt to fix problems you encounter, it would be appreciated if you could report them to us. And if you can take the time to describe what you were doing when you encountered the problem, it helps us fix the problems much more easily. You can report bugs using our template in our issue tracker on Github.

4.2 Continuous integration

As Microsoft provide generous levels of free usage to open source projects hosted on Github through their Azure DevOps service, ElectrumSV makes use of it for a range of purposes. Every time changes are pushed to Github, the following tasks are run:

- Unit tests on Windows, MacOS and Linux.
- Linting.
- Type checking.
- Code coverage analysis.
- Producing releases.

While Azure DevOps will do these things against each individual commit, we have configured the project to only do it against the latest commit.

4.2.1 Releases

There are two goals in having CI produce build files:

- We can use it to produce the build files we release publically.
- Members of the public can access and download build files for any build.

Using CI to produce official release files

By having CI produce the build files, this allows a developer to offload the processing work from their own computer and carry on working on other tasks. In addition there is some security in having the build files made within CI, where the CI obtains the source code directly from the latest commit on Github. And on generating the build files, also produces SHA256 hashes that can be used to validate the content at any later time.

Benefits of public build access

If a user is experiencing a bug, a developer can fix it and push the fix to Github. This will result in an automatic build on Azure DevOps, and if it succeeds will produce build files. The developer can point the user to the build, and although the user may not have an account with Azure DevOps they still have enough access that they can download build artifacts like the build files.

4.3 Release process

The process of making a release is as follows:

1. Write an article detailing the changes that are included in the release.
2. Put the article link in the correct places in the source code.
3. Update the version number in the correct places in the source code, including the `version.py` file.
4. Update the release date and time in the `version.py` file.
5. Tag the release.
6. Push the updated source code to the Github release branch.
7. Push the release tag to Github quickly before the CI starts the build. This will mark the release as stable, and cause the release files to be named by release version rather than git commit revision.
8. Verify that the CI is doing a stable build and wait for it to finish.
9. Ensure that all unit tests pass.
10. Ensure that linting and typing checks all pass.
11. Download all build files that are produced on Azure Devops as artefacts of the stable build.
12. Delete the installer `-setup.exe` executable for Windows.
13. Test that the built executables work on each of Windows and MacOS.
14. Generate GPG signatures for all build files.
15. Upload all the build files to the Amazon S3 storage.
16. Update the web site source code for the new build files and their download links.
17. Update the web site in general for the new version.

18. Update the web site `release.json` file for the new version, release date and time, and the signature using the release credentials signing key.
19. Publish the web site.
20. Publish the article detailing the changes included in the release.
21. Announce the release on Twitter, Metanet.ICU slack, Atlantistic Unwriter slack and anywhere else.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex