ElectrumSV

The ElectrumSV developers

Jan 02, 2023

GETTING STARTED

1 Getting started 3

2 Problem solving 33

3 Building on ElectrumSV 45

4 The ElectrumSV project 59

5 Indices and tables 69

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 know you have the official software and not malware?
Every person who had their coins stolen and was interested in investigating, identified that they had not downloaded from our official web site, and had instead obtained malware from some other fake site. Many swore they downloaded from the official site until their verified their download and found it to be malware. Read more about *verifying your download*.

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 Verifying your download

Probably a dozen people have reported having their coins stolen, and any who were willing to investigate found they had downloaded a malware version of ElectrumSV and not an official download. More than one asserted that they had downloaded from our web site, but what they meant was that they had downloaded from a fake web site that had stolen our design.

Downloading an executable from a web site and running it is risky, and what you are putting your trust in, is that because you download from our official web site you are getting an executable you can safely run. Despite this, well meaning people have downloaded from fake versions of our web site, and paid the price for it.

It is in your best interests to verify your download is the official one. The goal of this page is to try and show you how to do that.

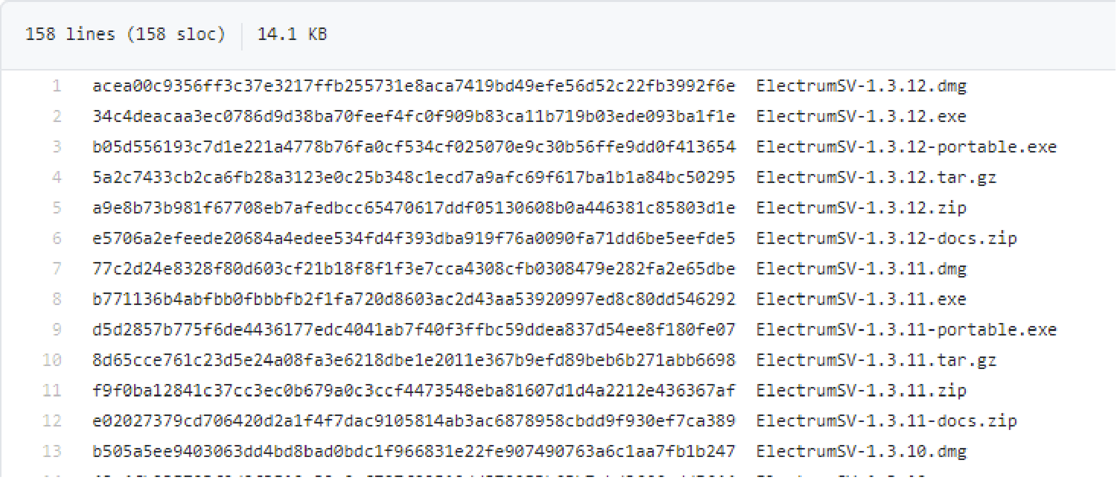
1.1.1 What are you verifying?

You will be checking the checksum (also known as a hash) of the file you downloaded. This is a standard algorithm that you can get lots of different software for, which will give you a series of letters and numbers that represent the uniqueness of your file. The algorithm we use for ElectrumSV is called SHA256 and we provide an official checksum for each file we make available. You will be comparing that official checksum to the one generated for your file. If it is the same, you should have the official version of that file. If it is different, you have downloaded malware instead.

The official checksums

We do not provide the checksums on the official web site where you find our download links, because this allows any attacker who manages to compromise the web site, to also replace the official checksums. Additionally, if there is a fake web site that offers both download links and checksums you should know something is fishy.

The official checksums are available from Github, where our open source code is located. You do not need to compare against the illustrative screenshot below, just click on the Github link and view them there.



```
158 lines (158 sloc) | 14.1 KB

acea00c9356ff3c37e3217ffb255731e8aca7419bd49efe56d52c22fb3992f6e  ElectrumSV-1.3.12.dmg
34c4deacaa3ec0786d9d38ba70feef4fc0f909b83ca11b719b03ede093ba1f1e  ElectrumSV-1.3.12.exe
b05d556193c7d1e221a4778b76fa0cf534cf025070e9c30b56ffe9dd0f413654  ElectrumSV-1.3.12-portable.exe
5a2c7433cb2ca6fb28a3123e0c25b348c1ecd7a9afc69f617ba1b1a84bc50295  ElectrumSV-1.3.12.tar.gz
a9e8b73b981f67708eb7afedbcc65470617ddf05130608b0a446381c85803d1e  ElectrumSV-1.3.12.zip
e5706a2efeede20684a4edee534fd4f393dba919f76a0090fa71dd6be5eefde5  ElectrumSV-1.3.12-docs.zip
77c2d24e8328f80d603cf21b18f8f1f3e7cca4308cfb0308479e282fa2e65dbe  ElectrumSV-1.3.11.dmg
b771136b4abfbb0fbbbfb2f1fa720d8603ac2d43aa53920997ed8c80dd546292  ElectrumSV-1.3.11.exe
d5d2857b775f6de4436177edc4041ab7f40f3ffbc59ddea837d54ee8f180fe07  ElectrumSV-1.3.11-portable.exe
8d65cce761c23d5e24a08fa3e6218dbe1e2011e367b9efd89beb6b271abb6698  ElectrumSV-1.3.11.tar.gz
f9f0ba12841c37cc3ec0b679a0c3ccf4473548eba81607d1d4a2212e436367af  ElectrumSV-1.3.11.zip
e02027379cd706420d2a1f4f7dac9105814ab3ac6878958cbdd9f930ef7ca389  ElectrumSV-1.3.11-docs.zip
b505a5ee9403063dd4bd8bad0bdc1f966831e22fe907490763a6c1aa7fb1b247  ElectrumSV-1.3.10.dmg
```

Fig. 1: The list SHA256 hashes for the official downloads.

1.1.2 Verifying your download

There is no easy way to check a download. Some level of technical competence is useful, although if you do not consider yourself technically competent and can follow instructions you should still be able to do it. Others have managed to do it, and as we get these instructions into a more approachable state over time, you should be able to as well.

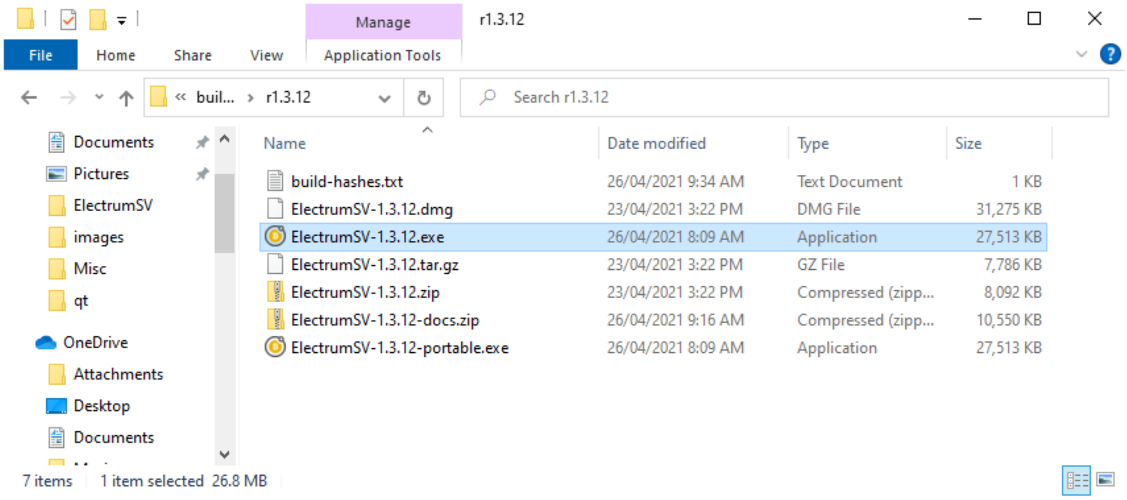
Find your operating system below, and check out the options listed for it. Some of them may be better than others, but some assurance that your download is legitimate is better than nothing.

Windows

Several methods of verifying your download on Windows are provided below. Any one should be good enough, but if you are a user who primarily uses a web browser you may need to learn to use the explorer or console.

Using the digital signatures

Thanks to the kindness of the Bitcoin Association, we now have the ability to sign our Windows executables from version 1.3.12 and above. In theory the presence of our signature on the executable you downloaded should be just as reliable as checking the checksum. You can check if the executable you downloaded has our signature, and if it is present you can assume that the file should be legitimate. Your first step is to find the executable you downloaded with the Windows explorer. You can open the Windows explorer with the `windows` and `e` key, then locate the directory your executable is located in.



Name	Date modified	Type	Size
build-hashes.txt	26/04/2021 9:34 AM	Text Document	1 KB
ElectrumSV-1.3.12.dmg	23/04/2021 3:22 PM	DMG File	31,275 KB
ElectrumSV-1.3.12.exe	26/04/2021 8:09 AM	Application	27,513 KB
ElectrumSV-1.3.12.tar.gz	23/04/2021 3:22 PM	GZ File	7,786 KB
ElectrumSV-1.3.12.zip	23/04/2021 3:22 PM	Compressed (zipp...	8,092 KB
ElectrumSV-1.3.12-docs.zip	26/04/2021 9:16 AM	Compressed (zipp...	10,550 KB
ElectrumSV-1.3.12-portable.exe	26/04/2021 8:09 AM	Application	27,513 KB

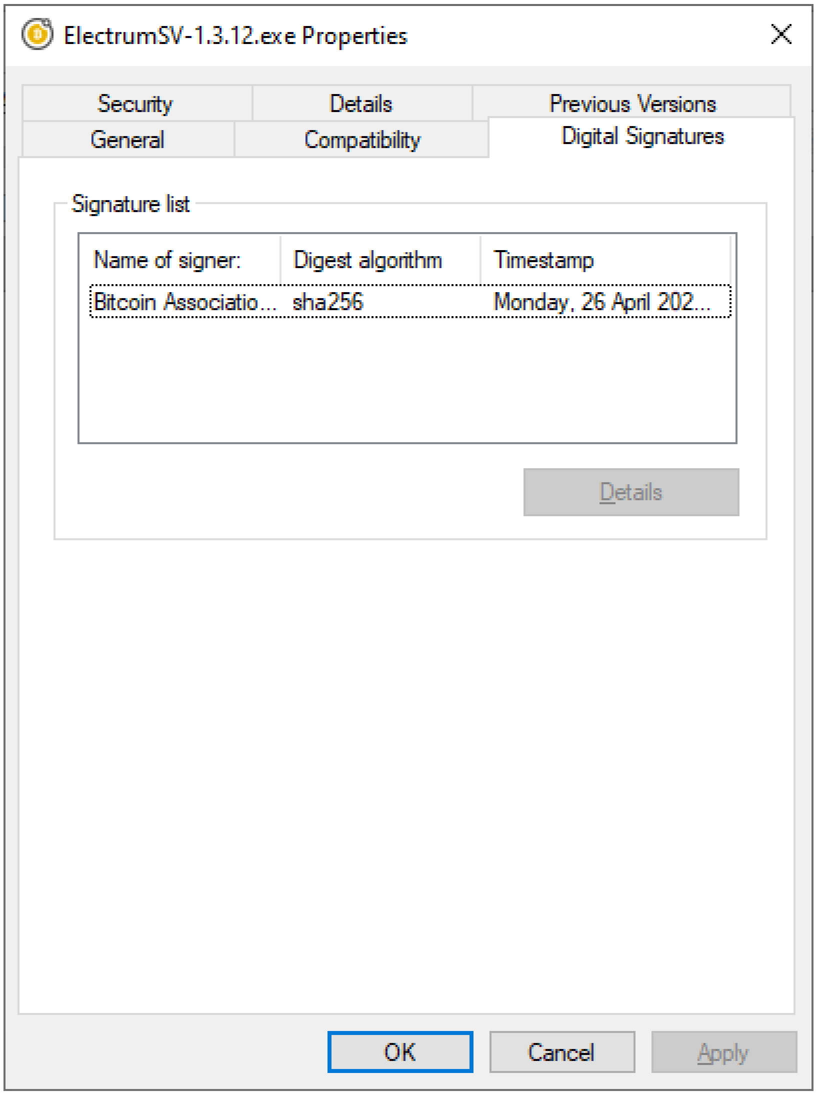
Fig. 2: Windows explorer.

Right click on the file, and select `Properties`. This should open the properties window for the file, where you should select the `Digital Signatures` tab to see the signature.

From there click on `Details` and then `View Certificate`. You should see a certificate with the following information for the given version.

1.3.12 and above

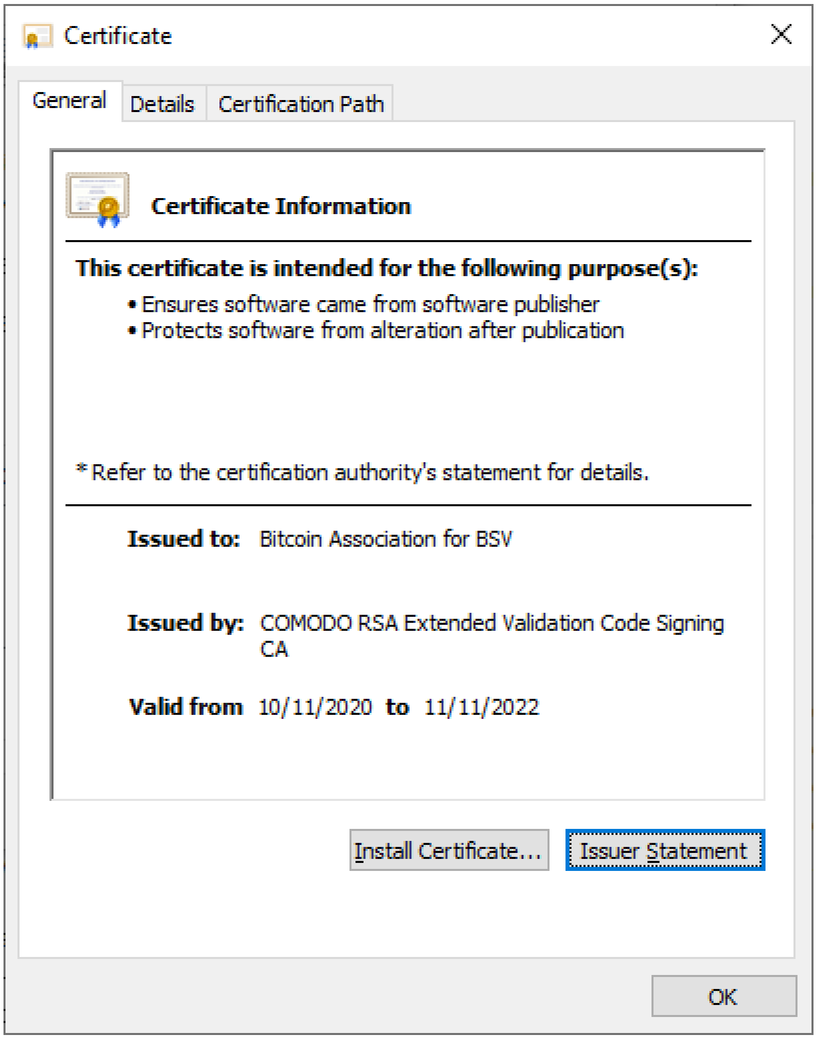
The certificate should be issued to `Bitcoin Association for BSV`, be issued by `COMODO RSA Extended Validation Code Signing CA` and as of the time of writing be valid from `10/11/2020` to `11/11/2022`.



ElectrumSV-1.3.12.exe Properties

Name of signer:	Digest algorithm	Timestamp
Bitcoin Associatio...	sha256	Monday, 26 April 202...

Fig. 3: The digital signature.



Certificate Information

This certificate is intended for the following purpose(s):

- Ensures software came from software publisher
- Protects software from alteration after publication

* Refer to the certification authority's statement for details.

Issued to: Bitcoin Association for BSV

Issued by: COMODO RSA Extended Validation Code Signing CA

Valid from 10/11/2020 **to** 11/11/2022

Fig. 4: The certificate the file was signed with.

Using certutil

`certutil` is already present in your Windows installation already. However, it requires opening a command prompt to run it, which might be something beyond some users. Press the *Windows* key and the `s` key at the same time, this will open the Windows searchy thing and there you can type `cmd` and then press the `enter` key to open a command prompt.

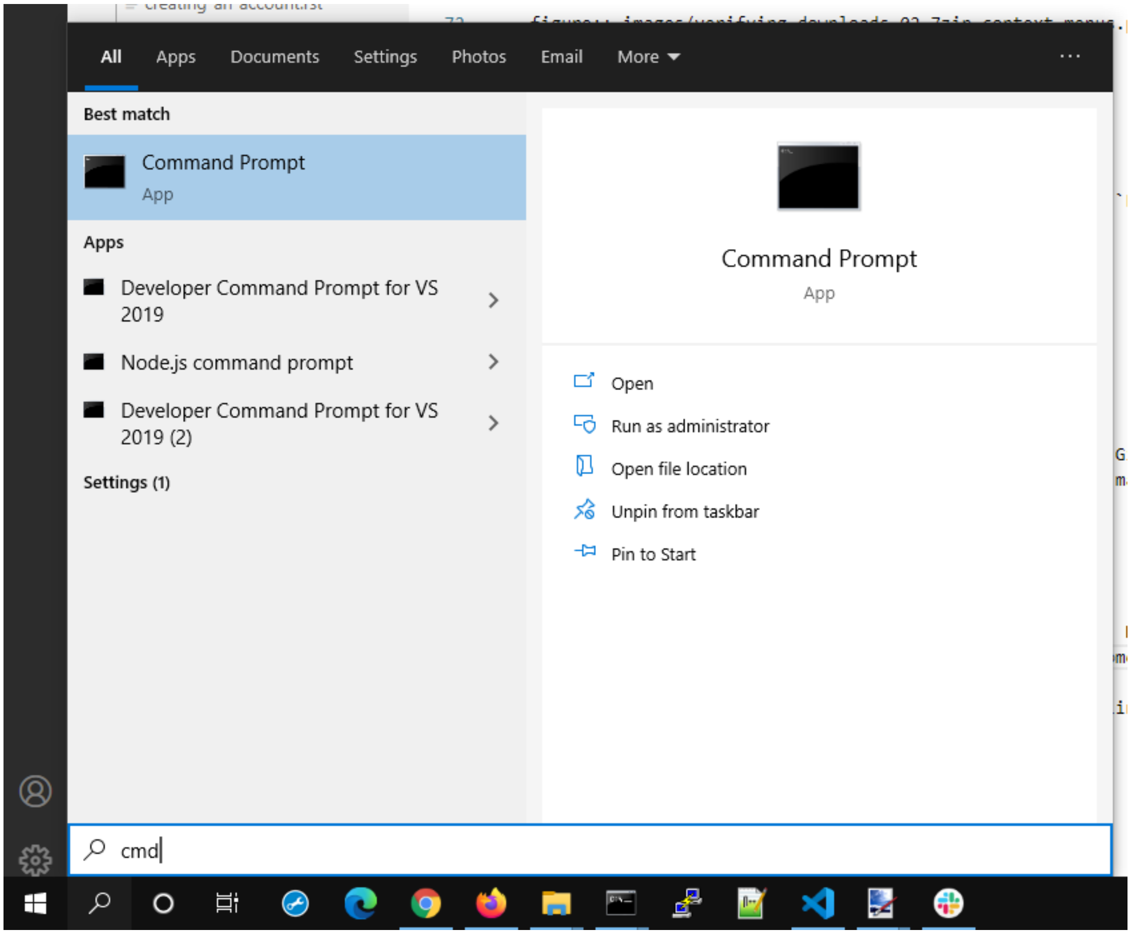
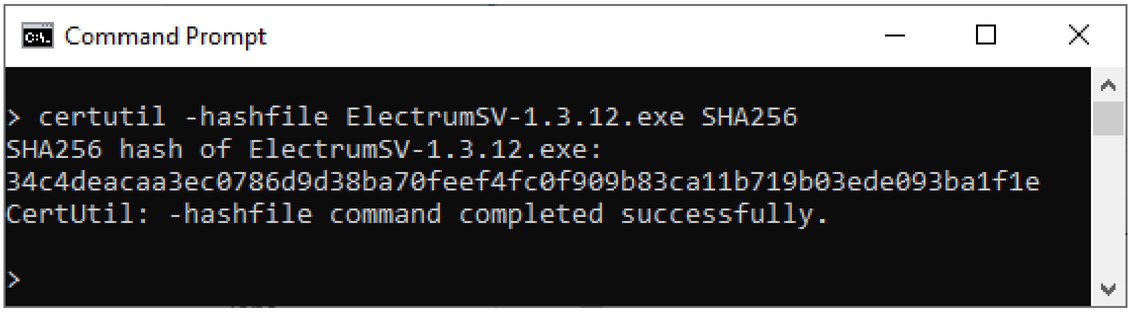


Fig. 5: Opening a command prompt.

Then you need to change the directory until you are in the same directory as the file you wish to get a checksum for. The `cd` command is used for this. Then you can use the certutil command to generate a SHA 256 checksum for that file. The syntax is `certutil --hashfile <filename> SHA256`, but remember you need to replace `<filename>` with the actual file name. You can see an illustration of this in the image below.

If you find the `ElectrumSV-1.3.12.exe` entry in the linked Github list, you can see it matches the certutil checksum result. The case of the letters does not matter, both lower case and upper case are equivalent. If you get a different result, and the command complains that it cannot find the file, then the file is not in the current directory. You need to use the `cd` command to change the current directory as mentioned above.



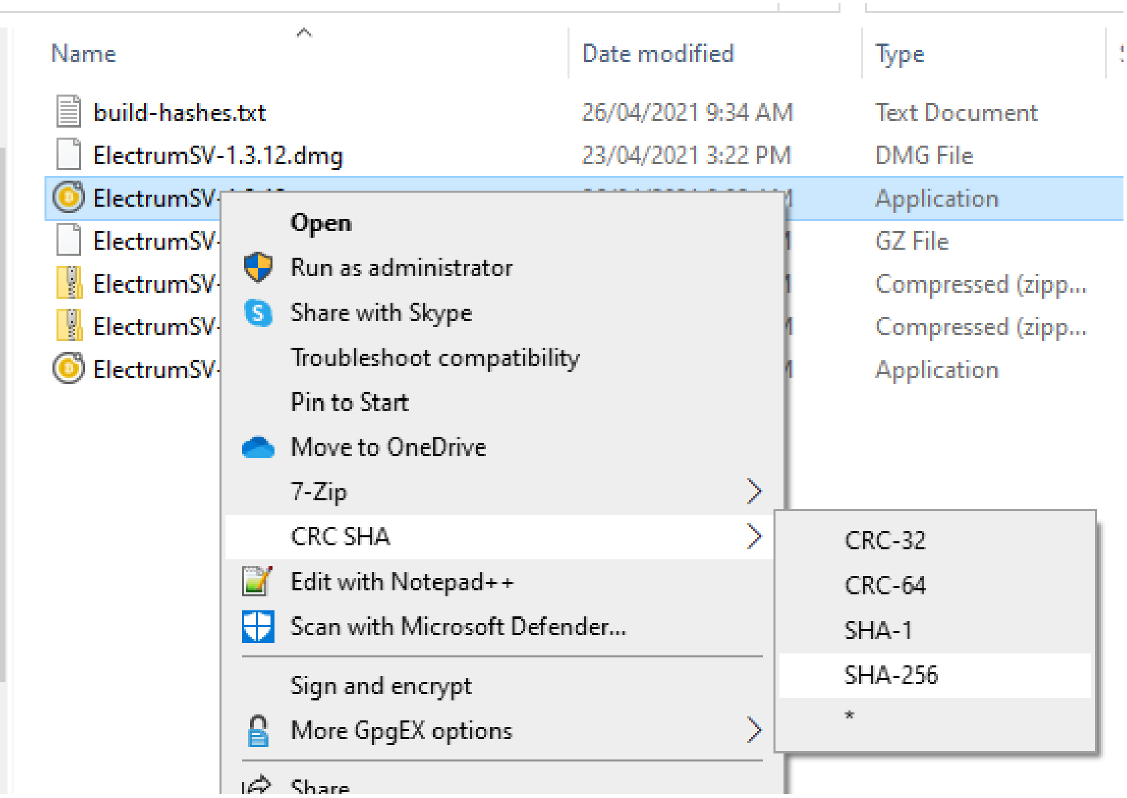
```
> certutil -hashfile ElectrumSV-1.3.12.exe SHA256
SHA256 hash of ElectrumSV-1.3.12.exe:
34c4deacaa3ec0786d9d38ba70feef4fc0f909b83ca11b719b03ede093ba1f1e
CertUtil: -hashfile command completed successfully.

>
```

Fig. 6: The certutil checksum result.

Using 7-Zip

This requires that you download the 7-Zip installer. Any of the non-standalone executables from the 7-Zip web site, should be fine. Download one and install it. Once it is installed, you should have a handy context menu available that can give you the SHA 256 checksum for your file. Simply select your file, open the context menu and generate the checksum. Do not reflect on the fact that no-one in their life ever wanted to "Share with Skype" and that they put it up the top before all the useful stuff.

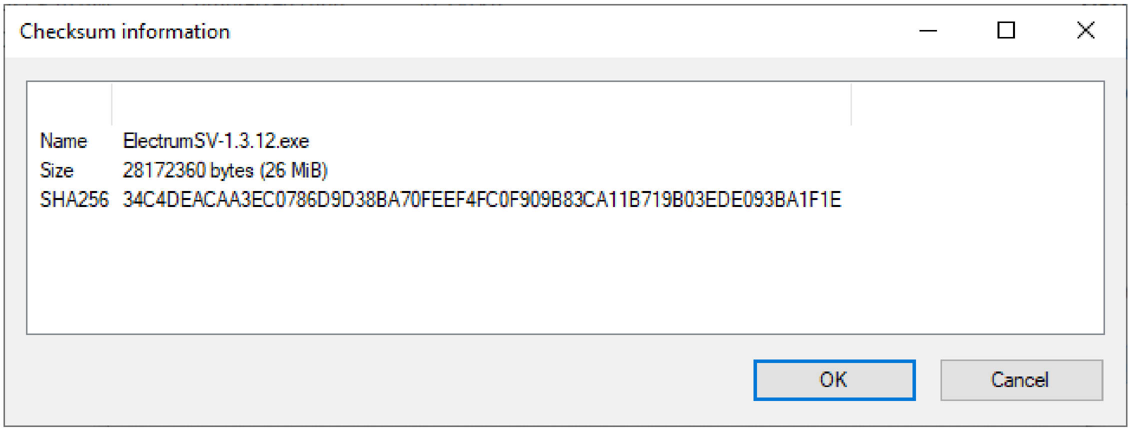


Name	Date modified	Type
build-hashes.txt	26/04/2021 9:34 AM	Text Document
ElectrumSV-1.3.12.dmg	23/04/2021 3:22 PM	DMG File
ElectrumSV-		Application
ElectrumSV-		GZ File
ElectrumSV-		Compressed (zipp...
ElectrumSV-		Compressed (zipp...
ElectrumSV-		Application

Fig. 7: The 7-Zip context menu.

In this case, we selected the `SHA-256` menu option for the `ElectrumSV-1.3.12.exe` file and the following image shows the resulting checksum.

If you find the `ElectrumSV-1.3.12.exe` entry in the linked Github list, you can see it matches the 7-zip checksum result. The case of the letters does not matter, both lower case and upper case are equivalent.



Name ElectrumSV-1.3.12.exe
Size 28172360 bytes (26 MiB)
SHA256 34C4DEACAA3EC0786D9D38BA70FEEF4FC0F909B83CA11B719B03EDE093BA1F1E

Fig. 8: The 7-Zip checksum result.

MacOS

The following approaches require the user to deal with the terminal. If you are unable to work out how to do this, remember you can always file a support request on the official ElectrumSV issue tracker.

shasum

This approach requires no application installation, but it does involve you being willing to use the `terminal` application. If you do not know how to locate this, start by opening the `launchpad` application using it's rocket icon in the dock.

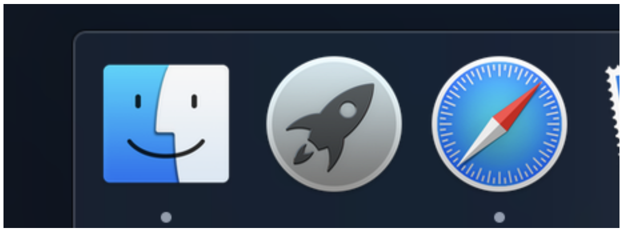


Fig. 9: Open the launchpad application search.

You should see the screen shown below. Enter `terminal` and it should show you one matching application which you should open.

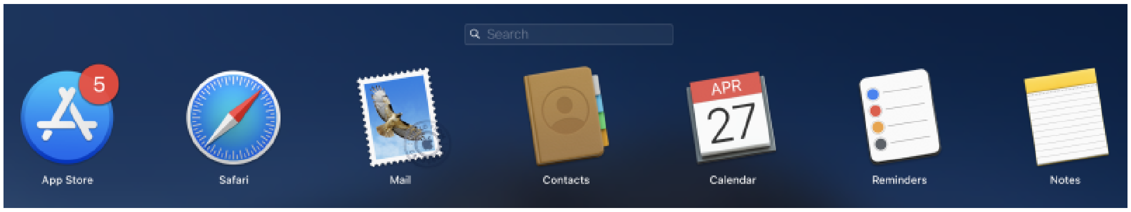
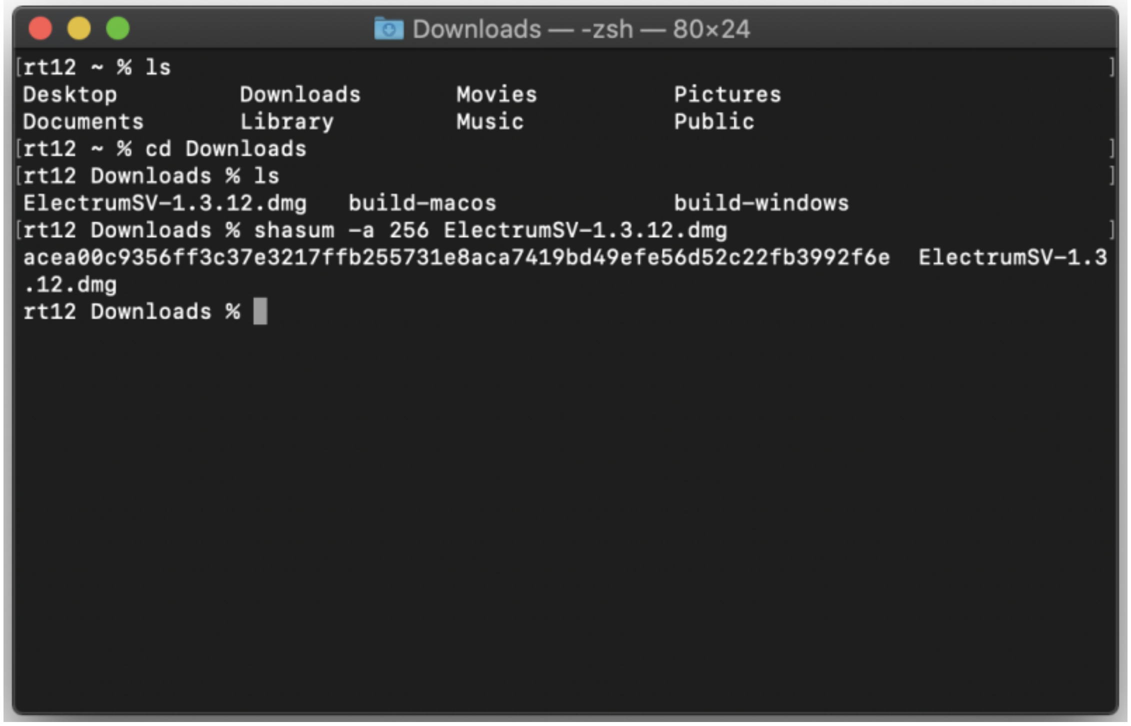


Fig. 10: Search for the 'terminal' application.

Work out what directory the terminal is looking at, and change it using the `cd` command. In the case shown below, the downloaded file was conveniently located in the `Downloads` folder and as this should also be the case for you the required commands should be the same. Type `cd Downloads` followed by `shasum -a 256 <filename>` where you replace <filename> with the actual file name of your download. Shown below, the file name was ElectrumSV-1.3.12.dmg and if you downloaded this file you also would use shasum -a 256 ElectrumSV-1.3.12.dmg as shown.



```
rt12 ~ % ls
Desktop         Downloads       Movies          Pictures
Documents       Library         Music           Public
rt12 ~ % cd Downloads
rt12 Downloads % ls
ElectrumSV-1.3.12.dmg   build-macos             build-windows
rt12 Downloads % shasum -a 256 ElectrumSV-1.3.12.dmg
acea00c9356ff3c37e3217ffb255731e8aca7419bd49efe56d52c22fb3992f6e  ElectrumSV-1.3
.12.dmg
rt12 Downloads % 
```

Fig. 11: Run the 'shasum' application on your downloaded file.

If you find the ElectrumSV-1.3.12.dmg entry in the linked Github list, you can see it matches the shasum checksum result. The case of the letters does not matter, both lower case and upper case are equivalent. If you get a different result, and the command complains that it cannot find the file, then the file is not in the current directory. You need to use the cd command to change the current directory as mentioned above.

GNU Privacy Guard

By installing GNU Privacy Guard (GPG) you have a way to verify that the signatures provided by the developers for the files you download, prove those files came from those developers. This is quite involved to do, but it might be that you are more comfortable with this approach.

Start by downloading and installing GPG from the GPGTools web site. This gives you a way to check signatures for files. The next step is to obtain the keys for the ElectrumSV developers, and to register them with GPG. This is a little complicated so you need to follow these steps.

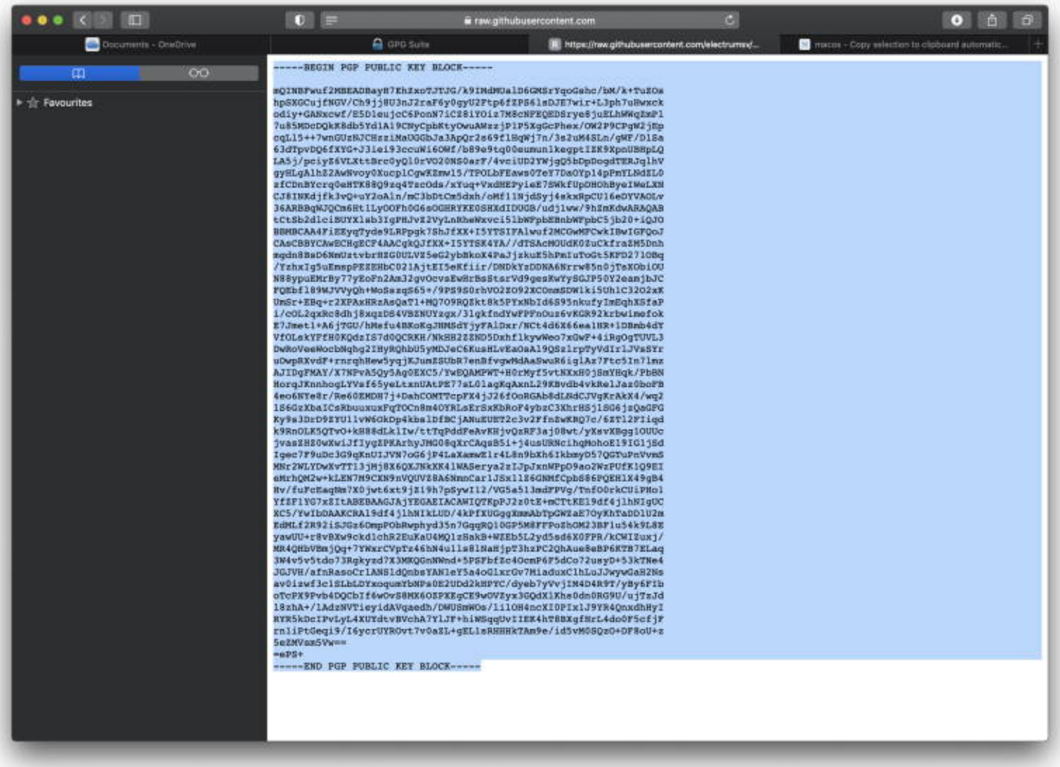
Open the pubkeys folder from the official ElectrumSV Github repository in Safari. You should see two files listed, rt121212121.asc and kyuupichain.asc. For each file perform the following key import actions.

Key import

Remember that this has to be done for all of the listed public keys in the ElectrumSV Github folder. Once you are viewing the raw page for a key, select (press Command with a) and copy (press Command and c) the key text.

As soon as you have copied the key text, the GPG application you installed will signal that it has detected a public key was copied. You will see it's icon in your dock jumping up and down. Click on it to import the key.

The GPG application will require you to approve the import, so go ahead and do that.


raw.githubusercontent.com
Documents - OneDrive
GPG Suite
https://raw.githubusercontent.com/electrumsv/...
macos - Copy selection to clipboard automatic...
Favourites

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBFwuf2MBEADBayH7EhZxoTJTJG/k9IMdMUalD6GMSrYqoGehc/bM/k+TuXOx
hpSXGCujfNGV/Ch9jj8U3nJ2raF6y0gyU2Ftp6fZPS6lsDJE7wir+L3ph7uHwxck
odiy+GANxcwf/E5DleujcC6PonN7iC28iYOiz7M8cNFEQEDSrye8juELhWWqZmPl
7u85MDcDQkK8db5Yd1Ai9CNyCpbKtyOwuAMzzjP1P5XgGcPhex/OW2P9CPgW2jEp
cqL15++7wnGUzHJCHzziMaUGGbJa3ApQr2s69f1HqWj7n/3s2uM4SLn/gWF/D1Sa
63dTpvDQ6fXYG+J3iei93ccuWi6OWf/b89e9tq00eumunlkegptIZK9XpnUBHpLQ
LA5j/pciyZ6VLXttBrcGyQl0rVO20NS0arF/4vciUD2YWjgQ5bDpDogdTERJqlhV
gyHLgAlhZ2AwNvoy0XucplCgwKZmw15/TPOLbFEaws0TeY7Da0Ypl4pPmYLNdZL0
zfCDnBYcrq0eHTK88Q9zq4TzcOds/xYuq+VxdHEPyieE7SWkfUpDHOhByeIWeLXN
CJ8INKdjfk3vQ+uY2oAln/mC3bDtCm5dxh/oMf11NjdSyj4skxRpCU16eOYVAOLv
36ARBBqWJQCm6Ht1LyOOFh0G6sOGHRYKE0SHXdIDUGB/udjlww/9hZmKdwARAQAB
tCtSb2dlciBUYXlab3IgPHJvZ2VyLnRheWxvci5lbWFpbEBnbWFpbC5jb20+iQJO
BBMBCAA4FiEEyqTyde9LRPpgk7ShJfXX+I5YTSIFAlwuf2MCGwMFCwkIBwIGFQoJ
CAsCBBYCAwECHgECF4AACgkQJfXX+I5YTSK4YA//dTSAcMOUdK0ZuCkfraZM5Dnh
mgdn8BaD6NmUztvbrHZG0ULVZ5eG2ybBkoX4PaJjzkuE5hPmIuToGt5KFD271OBq
/YzhxIg5uEmspPEZEHbC02lAjtEI5eKfiir/DNDkYzDDNA6Nrrw85n0jTeXObiOU
N88ypuEMrBy77yEoFn2Am32gvOcvsEwHrBsStsrVd9gesKwYySGJP50Y2eamjbJC
FQEbfl89MJVVyQh+WoSezqS65+/9PS9S0rhVO2ZO92XCOmmSDWlki5Uh1C32O2xK
UmSr+EBq+r2XPAxHRzAsQaT1+MQ7O9RQZkt8k5PYxNbId6S95nkufyImEqhXSfaP
i/cOL2qxRc8dhj8xqzD84VBZNUYzgx/3lqkfndYwPPFnOuz6vKGR92krbwimefok
E7Jmet1+A6jTGU/hMefu4BKoKgJHMSdYjyFAlDxr/NCt4d6X66ea1HR+1DBmb4dY
VfOLskYPfH0KQdzIS7d0QCRKH/NkHH2ZZND5DxhflkywWeo7xGwF+4iRgOgTUVL3
DwRoVeeWocbNqbg2IHyRQhbU5yMDJeC6KusHLvEaOaAl9QSzlrpTyVdIrlJVsSYr
uDwpRXvdF+rnrqhHew5yqjKJumZSUbR7enBfvgwMdAaSwuR6iglAx7Ftc5In7lmx
AJIDgFMAY/X7NPvA5Qy5Ag0EXC5/YwEQAMPWT+H0rMyf5vtNXxH0jSmYHqk/PbBN
HorqJKnnhogLYVsf65yeLtxnUAtPE77sL0lagKqAxnL29KBvdb4vkRelJaz0boFB
4eo6NYe8r/Re60EMDH7j+DahCOMTTcpFX4jJ26fOoRGAb8dLNdCJVgKrAkX4/wqJ
l86GzXbaICsRbuuxuxFqTOCn8m4OYRLsErSxKbRoF4ybzC3XhrHSjlSG6jzQaGFG
Ky9s3DrD9ZYUllvW6OkDp4kbalDfBCjANuZUET2c3v2FfnZwKRQ7c/6ZTl2FIiqd
k9RnOLK5QTvO+kH88dLklIw/ttTqPddFeAvKHjvQzRF3aj08wt/yXsvXBgg1OUUc
jvasZH20wXwiJflygZPKArhyJMG08qXrCAqsB5i+j4usURNcihqMohoEl9IGljSd
Igec7F9uDc3G9qKnUIJVN7oG6jP4LaXamwZlr4L8n9bXh6IkbmyD57QGTuPnVvmS
MNr2WLYDwXvTT13jMj8X6QXJNkXK4lWASerya2zIJpJxnWPpD9ao2WzPUfK1Q9EI
eMrhQM2w+kLEN7M9CXN9nVQUVZ8A6NmnCarlJSxllZ6GNMfCpb886PQEHlX49gB4
Hv/fuFcEaqNm7X0jwt6xt9jZi9h7pSywIl2/VG5a51JmdFPVg/TnfO0rkCUiPHol
YfZF1YG7xZItABEBAAGJAjYEGAEIACAWIQTKpPJ2z0tE+mCTtKEl9df4jlhNIgUC
XC5/YwIbDAAKCRAl9df4jlhNIkLUD/4kPfXUGgqXmmAbTpGWZaE7OyKhTaDDlU2m
EdMLf2R92iSJGz6OmpPObRwphyd35n7GqqRQl0GP5M8FFPoZhOM23BF1u54k9L8Z
yawUU+r8vBXw9ckdlchR2EuKaU4MQlzHakB+WZEb5L2yd5sd6X0FPR/kCWIZuxj/
MR4QHbVBmjQq+7YWxrCVpTz46hN4ulls8lNaHjpT3hzPC2QhAue8eBP6KTB7ELaq
3W4v5v5tdo73Rgkyzd7X3MKQGnNWnd+5PSFbfZc4OcmP6F5dCo72usyD+53kTNe4
JGJVH/afnRasoCrlANSldQnbsYANleY5a4oGlxrGv7MiaduxClhLuJJwywGaH2Ns
av0izwf3clSLbLDYxoqumYbNPs0E2UDd2kHPYC/dyeb7yVvjIM4D4R9T/yBy6FIb
oTcPX9Pvb4DQCbIf6wOvS8MX6OZPXEgCE9wOVZyx3GQdXlKhs0dn0RG9U/ujTzJd
l8zhA+/lAdzNVTieyidAVqaedh/DWUSmWOs/1ilOH4ncXI0PIxlJ9YR4QnxdhHyI
RYR5kDcIPvLyL4XUYdtvBVchA7YlJF+hiWSqqUvIIEK4hT8BXgfNrL4doOF5cfjF
rnliPtGeqi9/I6ycrUYROvt7v0aZL+gEL1sRHHHkTAm9e/id5vM0SQzO+DF8oU+z
5eZMVsm5Vw==
=ePS+
-----END PGP PUBLIC KEY BLOCK-----
```

Fig. 12: Select and copy the public key text.

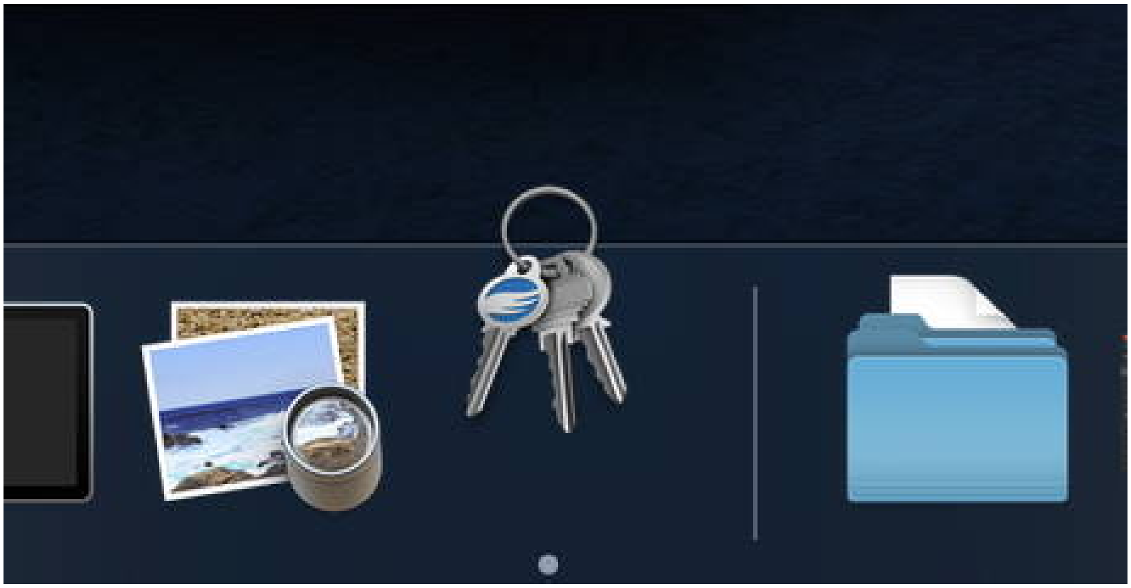
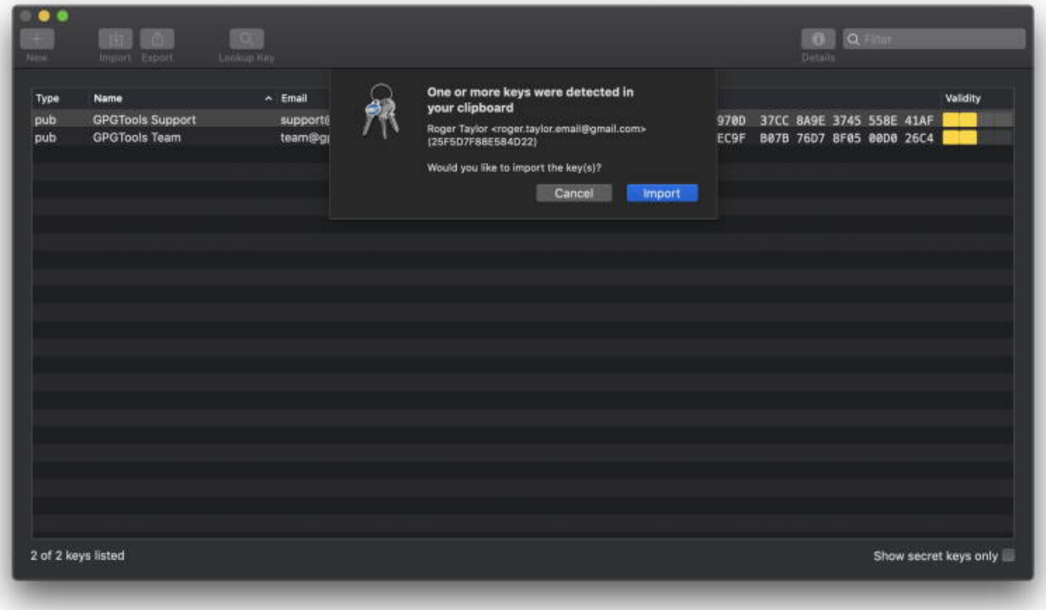


Fig. 13: Observe the GPG icon in the dock indicating that it can act on the copied key.



Type	Name	Email		Validity
pub	GPGTools Support	support(	970D 37CC 8A9E 3745 558E 41AF	
pub	GPGTools Team	team@g	EC9F B07B 76D7 8F05 00D0 26C4	

One or more keys were detected in your clipboard

Roger Taylor <roger.taylor.email@gmail.com>
(25F5D7F88E584D22)

Would you like to import the key(s)?

2 of 2 keys listed

Show secret keys only □

Fig. 14: Approve the public key import.

Once the public key is imported, you will see another sheet drop down to tell you if it was imported successfully or not. It will of course be successful.

You can confirm the key was imported successfully, by observing that it is now present in the list in the GPG application.

Go ahead and import any keys you haven't imported already, then you are all set to verify the signature of an ElectrumSV download when you need to.

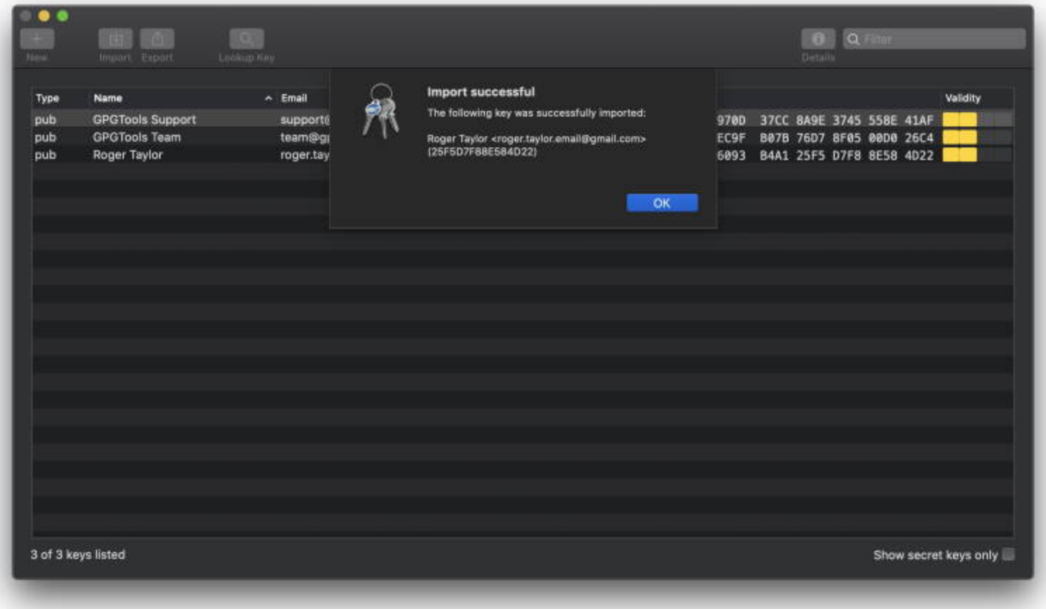
Verify a download

Let's say you have downloaded `ElectrumSV-1.3.12.dmg` from the official ElectrumSV downloads page. You now need to find and download the signature for that file, so that you can verify it was created by the ElectrumSV developers. The signatures are located on the official ElectrumSV web site, under it's download folder. The `.dmg` you downloaded was for version `1.3.12` so locate the folder by that name, and look inside it. You should see the signature file `ElectrumSV-1.3.12.dmg.sig`, which is what you need to download `ElectrumSV-1.3.12.dmg`.

Open the context menu for the `ElectrumSV-1.3.12.dmg` file (press `Control` when you click on the file). You will see a `Services` sub-menu, with an additional `OpenPGP: Verify Signature of File` beneath it. Click on this verify sub-menu.

The GPG application will verify the `.dmg` using the detected matching `.dmg.sig` file and let you know the result.

As you can see the signature was verified. If you want to go through the process of trusting the ElectrumSV keys, there is a link there you can use for next time.



Type	Name	Email		Validity
pub	GPGTools Support	support@	970D 37CC 8A9E 3745 558E 41AF	🟨🟨
pub	GPGTools Team	team@g	EC9F B07B 76D7 8F05 00D0 26C4	🟨🟨
pub	Roger Taylor	roger.tay	6093 B4A1 25F5 D7F8 8E58 4D22	🟨🟨

Import successful

The following key was successfully imported:

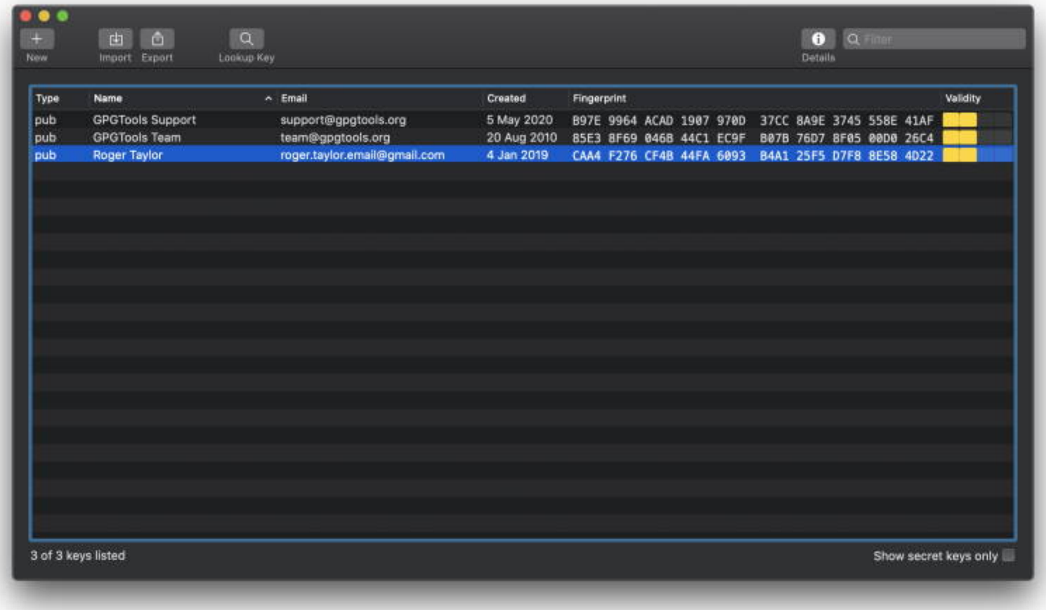
Roger Taylor <roger.taylor.email@gmail.com>
(25F5D7F88E584D22)

OK

3 of 3 keys listed

Show secret keys only □

Fig. 15: Observe the successful public key import.

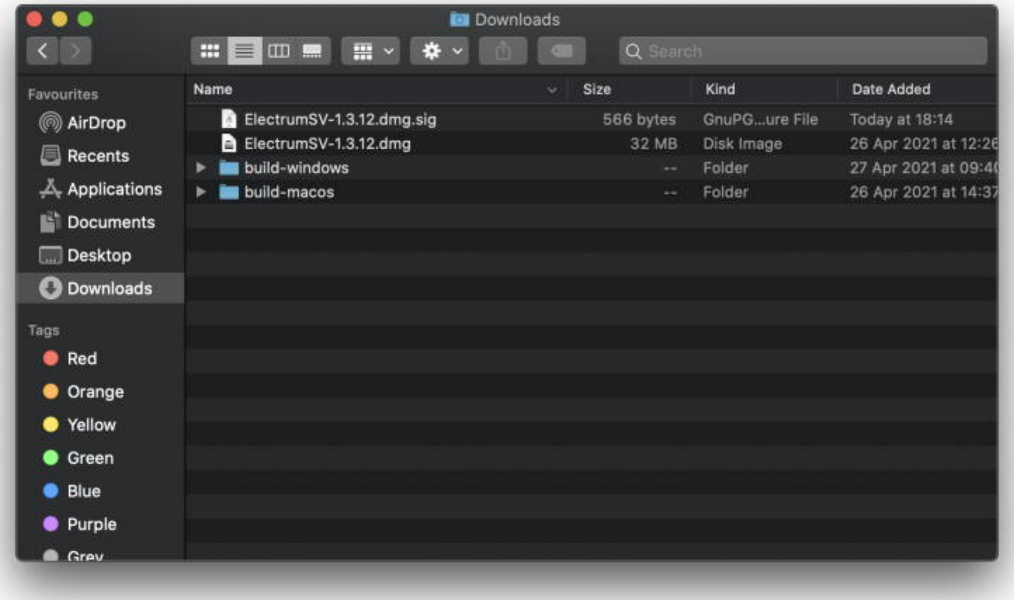


Type	Name	Email	Created	Fingerprint	Validity
pub	GPGTools Support	support@gpgtools.org	5 May 2020	B97E 9964 ACAD 1907 970D 37CC 8A9E 3745 558E 41AF	🟨🟨
pub	GPGTools Team	team@gpgtools.org	20 Aug 2010	85E3 8F69 046B 44C1 EC9F B07B 76D7 8F05 00D0 26C4	🟨🟨
pub	Roger Taylor	roger.taylor.email@gmail.com	4 Jan 2019	CAA4 F276 CF4B 44FA 6093 B4A1 25F5 D7F8 8E58 4D22	🟨🟨

3 of 3 keys listed

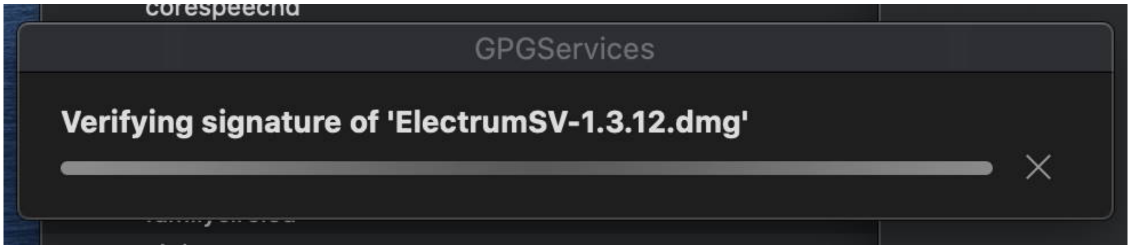
Show secret keys only □

Fig. 16: Observe the imported public key is present in your GPG application.



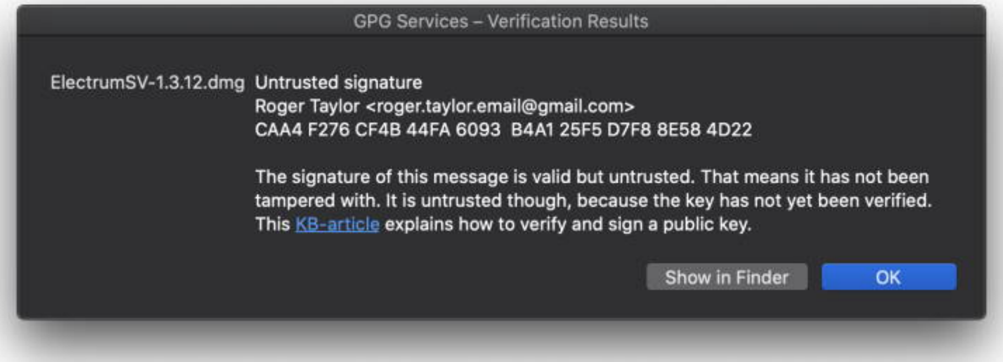
Name	Size	Kind	Date Added
ElectrumSV-1.3.12.dmg.sig	566 bytes	GnuPG...ure File	Today at 18:14
ElectrumSV-1.3.12.dmg	32 MB	Disk Image	26 Apr 2021 at 12:26
build-windows	--	Folder	27 Apr 2021 at 09:40
build-macos	--	Folder	26 Apr 2021 at 14:37

Fig. 17: Confirm you have downloaded both the `.dmg` and the matching `.dmg.sig` files.



Verifying signature of 'ElectrumSV-1.3.12.dmg'

Fig. 18: Open the context menu and select the OpenPGP verify entry.



ElectrumSV-1.3.12.dmg Untrusted signature
Roger Taylor <roger.taylor.email@gmail.com>
CAA4 F276 CF4B 44FA 6093 B4A1 25F5 D7F8 8E58 4D22

The signature of this message is valid but untrusted. That means it has not been tampered with. It is untrusted though, because the key has not yet been verified. This KB-article explains how to verify and sign a public key.

Fig. 19: Observe the verification result.

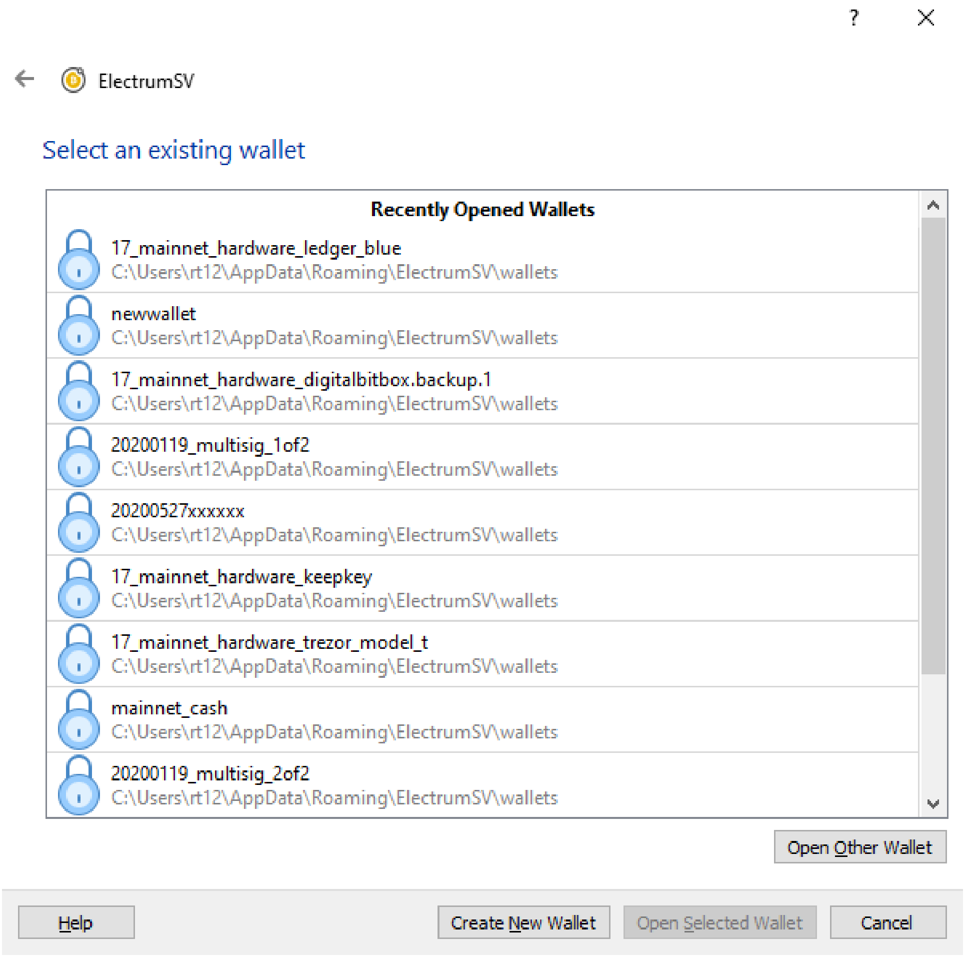
1.2 Creating a wallet

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.

1.2.1 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.



17_mainnet_hardware_ledger_blue
C:\Users\rt12\AppData\Roaming\ElectrumSV\wallets

newwallet
C:\Users\rt12\AppData\Roaming\ElectrumSV\wallets

17_mainnet_hardware_digitalbitbox.backup.1
C:\Users\rt12\AppData\Roaming\ElectrumSV\wallets

20200119_multisig_1of2
C:\Users\rt12\AppData\Roaming\ElectrumSV\wallets

20200527xxxxxx
C:\Users\rt12\AppData\Roaming\ElectrumSV\wallets

17_mainnet_hardware_keepkey
C:\Users\rt12\AppData\Roaming\ElectrumSV\wallets

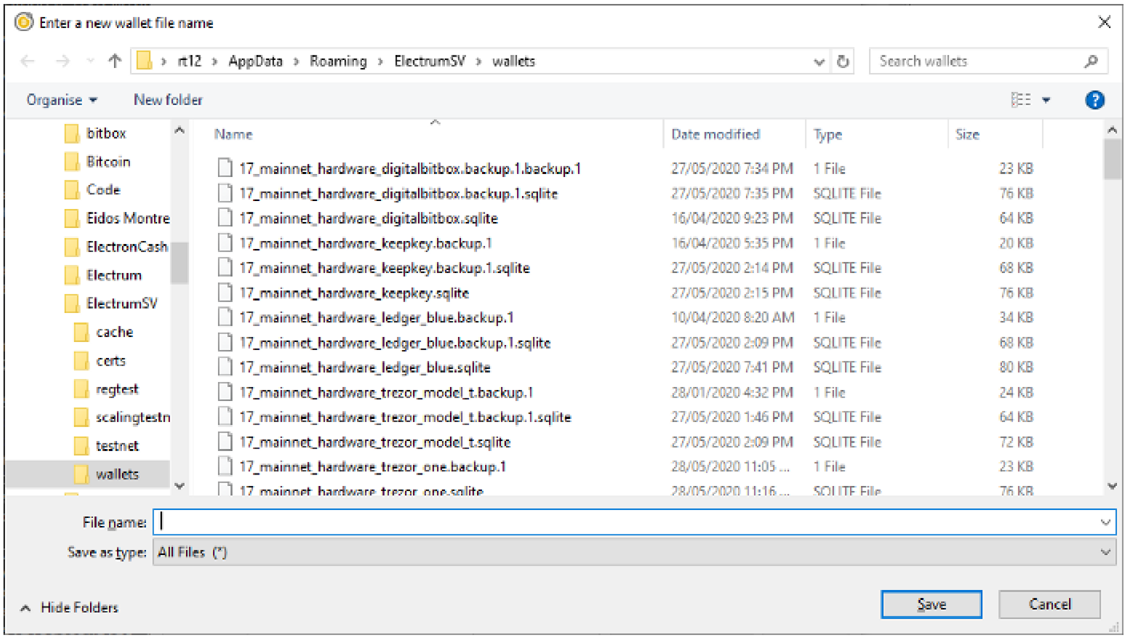
17_mainnet_hardware_trezor_model_t
C:\Users\rt12\AppData\Roaming\ElectrumSV\wallets

mainnet_cash
C:\Users\rt12\AppData\Roaming\ElectrumSV\wallets

20200119_multisig_2of2
C:\Users\rt12\AppData\Roaming\ElectrumSV\wallets

Fig. 20: 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).



Name	Date modified	Type	Size
17_mainnet_hardware_digitalbitbox.backup.1.backup.1	27/05/2020 7:34 PM	1 File	23 KB
17_mainnet_hardware_digitalbitbox.backup.1.sqlite	27/05/2020 7:35 PM	SQLITE File	76 KB
17_mainnet_hardware_digitalbitbox.sqlite	16/04/2020 9:23 PM	SQLITE File	64 KB
17_mainnet_hardware_keepkey.backup.1	16/04/2020 5:35 PM	1 File	20 KB
17_mainnet_hardware_keepkey.backup.1.sqlite	27/05/2020 2:14 PM	SQLITE File	68 KB
17_mainnet_hardware_keepkey.sqlite	27/05/2020 2:15 PM	SQLITE File	76 KB
17_mainnet_hardware_ledger_blue.backup.1	10/04/2020 8:20 AM	1 File	34 KB
17_mainnet_hardware_ledger_blue.backup.1.sqlite	27/05/2020 2:09 PM	SQLITE File	68 KB
17_mainnet_hardware_ledger_blue.sqlite	27/05/2020 7:41 PM	SQLITE File	80 KB
17_mainnet_hardware_trezor_model_t.backup.1	28/01/2020 4:32 PM	1 File	24 KB
17_mainnet_hardware_trezor_model_t.backup.1.sqlite	27/05/2020 1:46 PM	SQLITE File	64 KB
17_mainnet_hardware_trezor_model_t.sqlite	27/05/2020 2:09 PM	SQLITE File	72 KB
17_mainnet_hardware_trezor_one.backup.1	28/05/2020 11:05 ...	1 File	23 KB
17_mainnet_hardware_trezor_one.sqlite	28/05/2020 11:16 ...	SQLITE File	76 KB

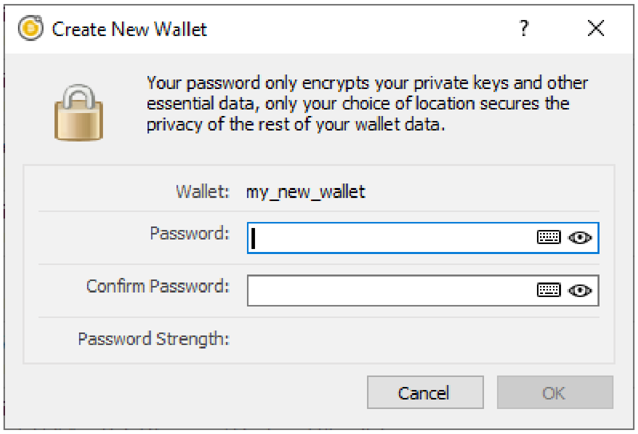
File name:

Save as type: All Files (*)

Fig. 21: The wallet file name dialog.

1.2.2 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.



Your password only encrypts your private keys and other essential data, only your choice of location secures the privacy of the rest of your wallet data.

Wallet: my_new_wallet

Password:

Confirm Password:

Password Strength:

Fig. 22: The wallet file name dialog.

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.

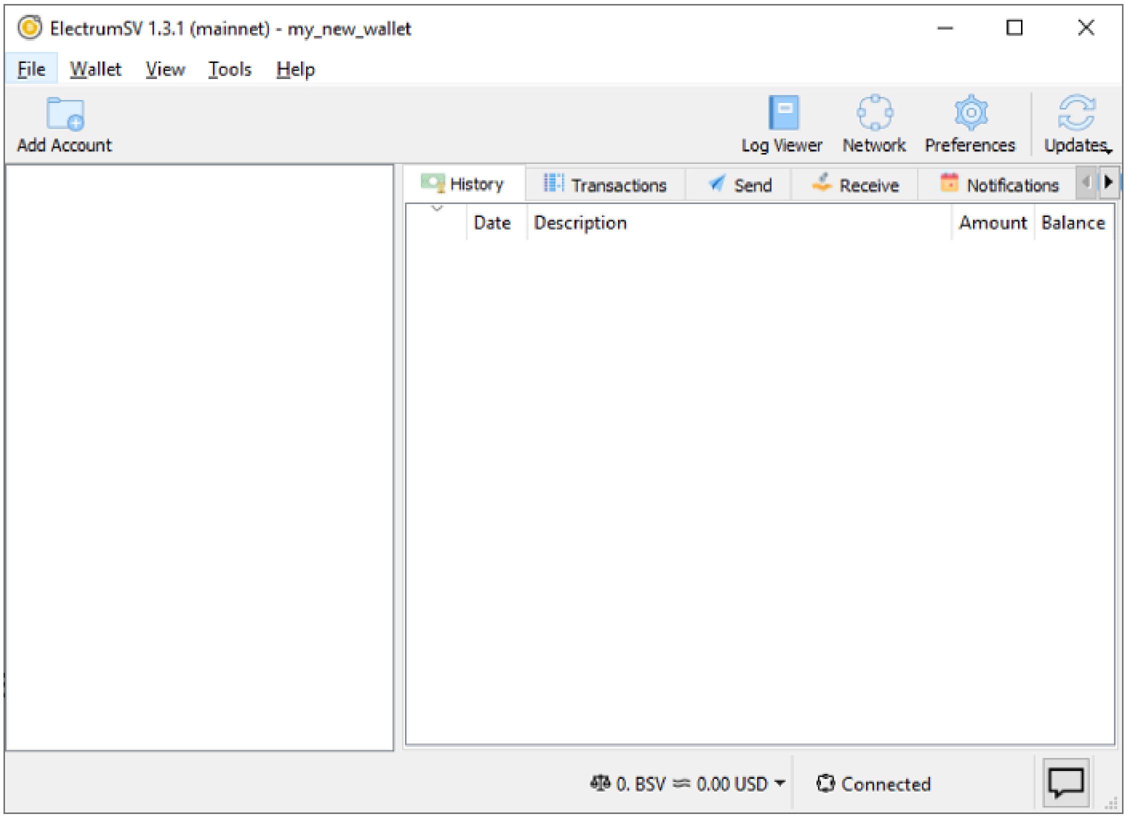
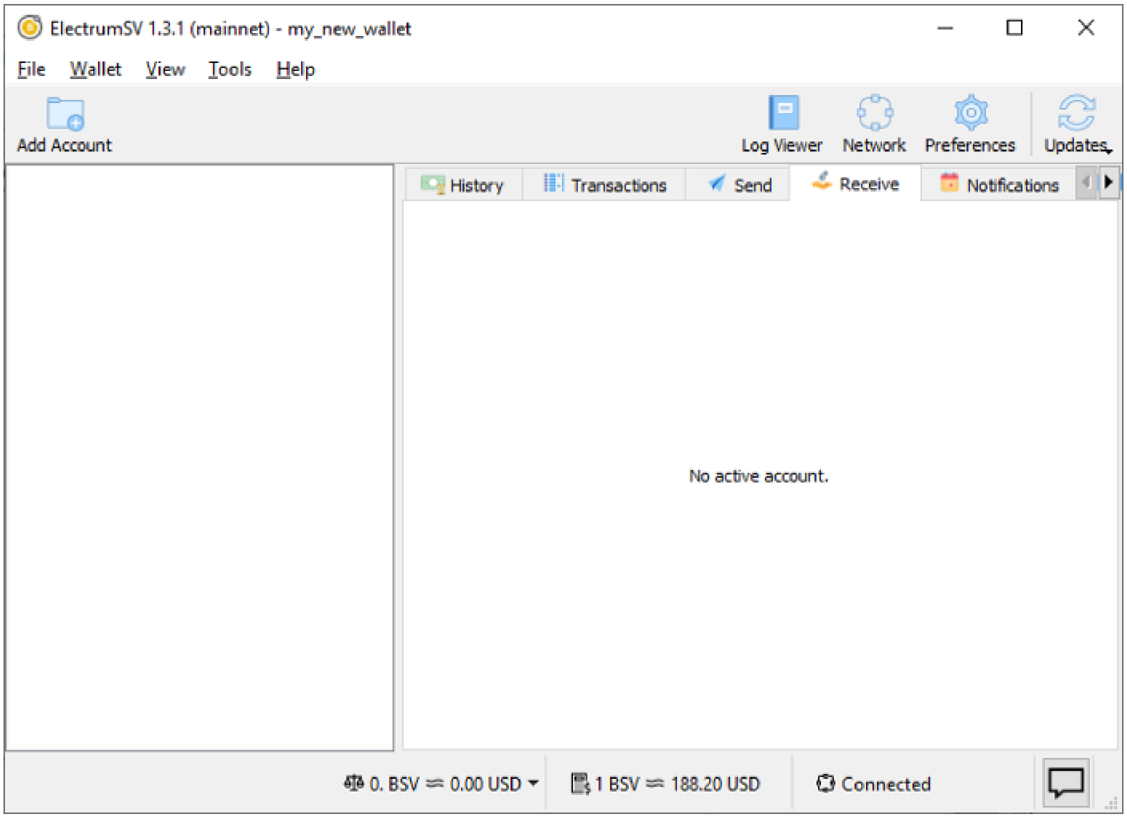


Fig. 23: The new wallet's wallet window.



No active account.

Fig. 24: The receiving tab is disabled.

1.3 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.

This guide solely covers creating a "Standard" account.

1.3.1 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.

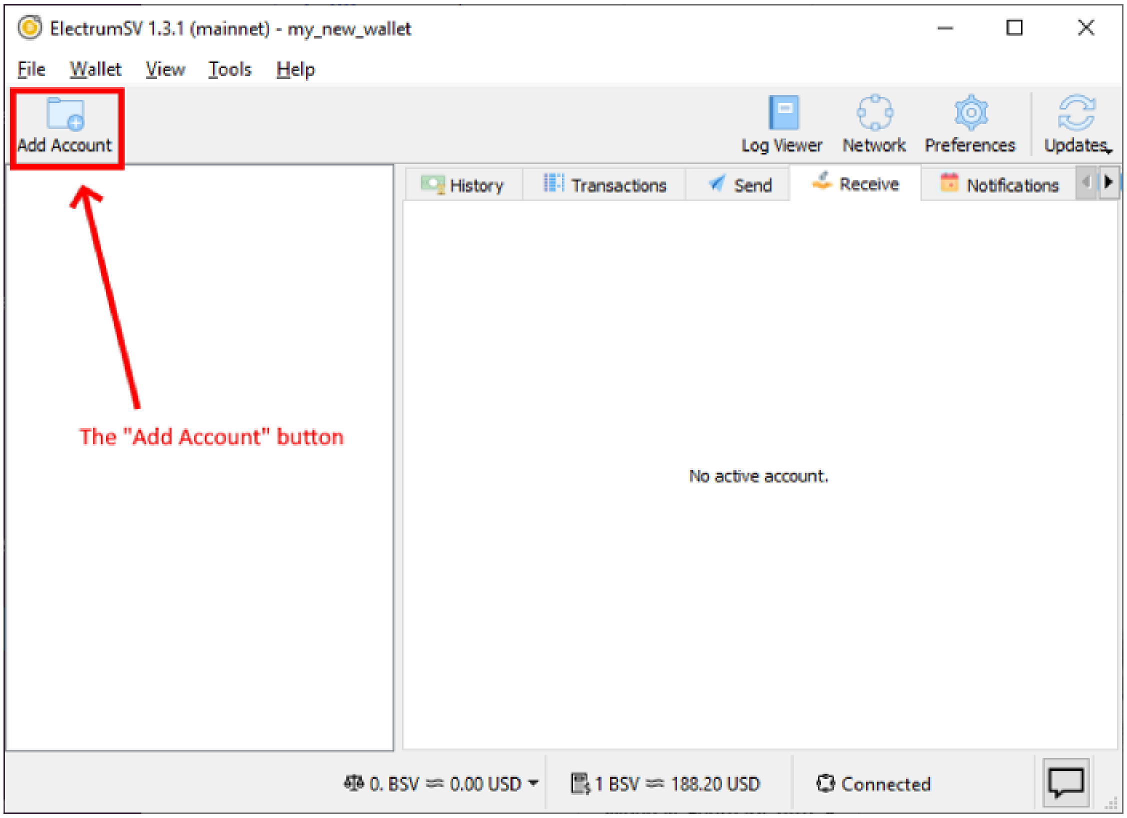
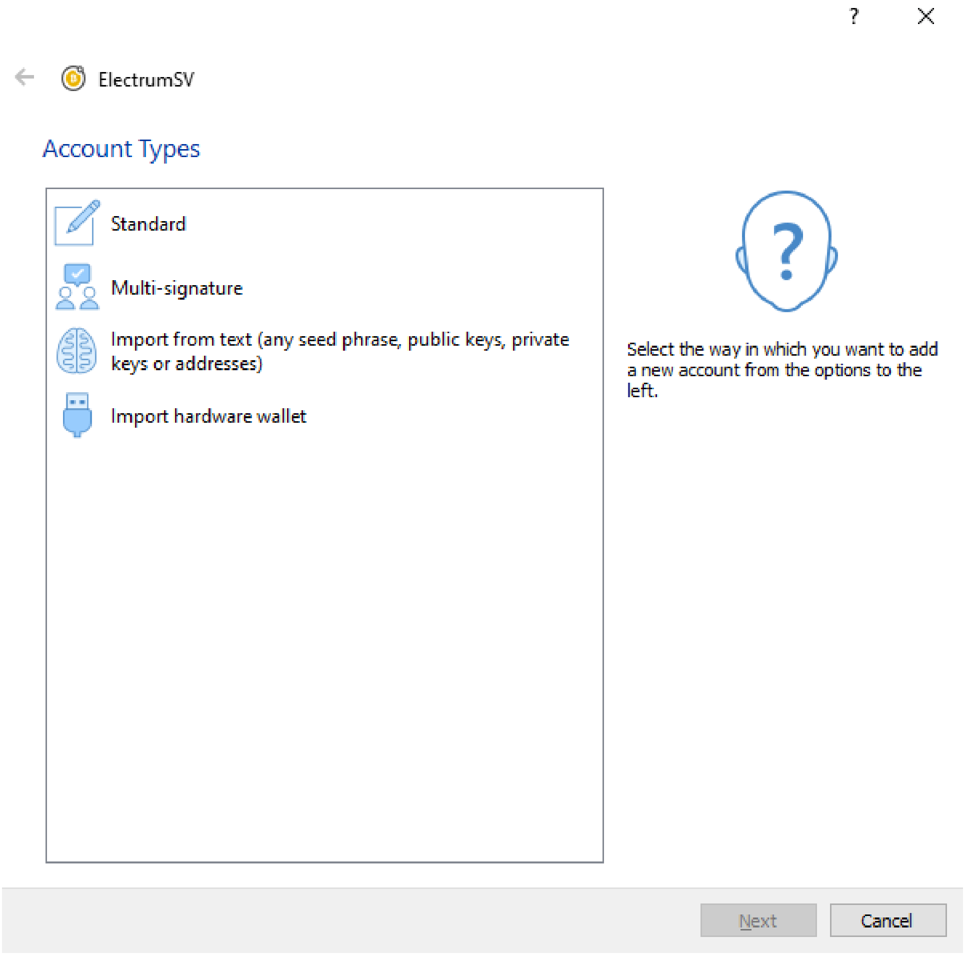


Fig. 25: The "Add Account" button highlighted.

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



Account Types

Standard

Multi-signature

Import from text (any seed phrase, public keys, private keys or addresses)

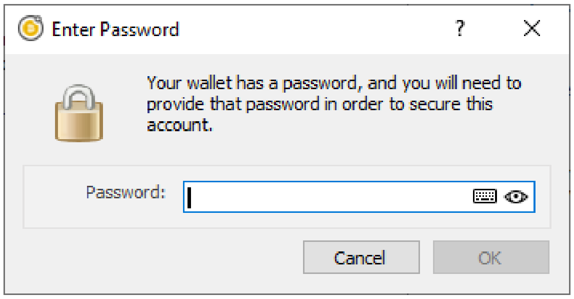
Import hardware wallet

Select the way in which you want to add a new account from the options to the left.

Fig. 26: The Account Types page in the Account Wizard.

1.3.2 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.



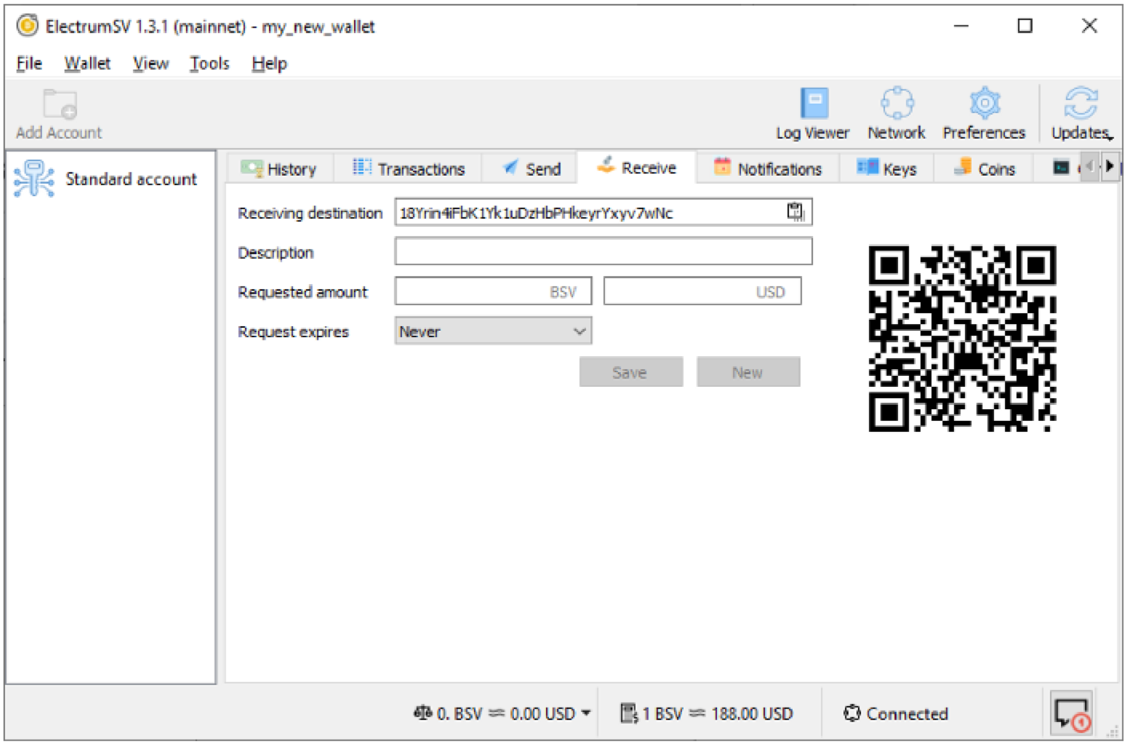
Enter Password

Your wallet has a password, and you will need to provide that password in order to secure this account.

Password:

Fig. 27: The password dialog.

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.



ElectrumSV 1.3.1 (mainnet) - my_new_wallet

Standard account

Receiving destination: 18Yrin4iFbK1Yk1uDzHbPHkeyrYxyv7wNc

Description:

Requested amount: BSV USD

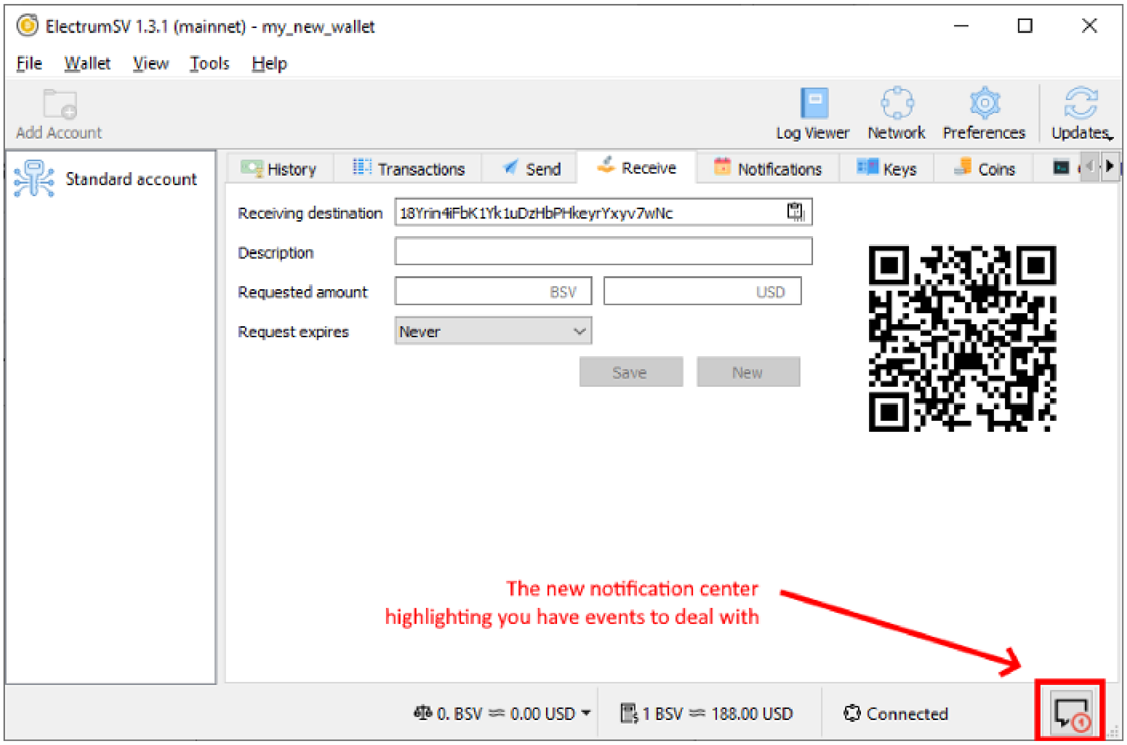
Request expires: Never

0. BSV ≈ 0.00 USD | 1 BSV ≈ 188.00 USD | Connected

Fig. 28: The new account's receiving tab.

1.3.3 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.



Receiving destination 18Yrin4iFbK1Yk1uDzHbPHkeyrYxyv7wNc

Description

Requested amount BSV USD

Request expires Never

The new notification center
highlighting you have events to deal with

0. BSV ≈ 0.00 USD
1 BSV ≈ 188.00 USD
Connected

Fig. 29: The wallet's notifications indicator.

1.3.4 The initial backup notification

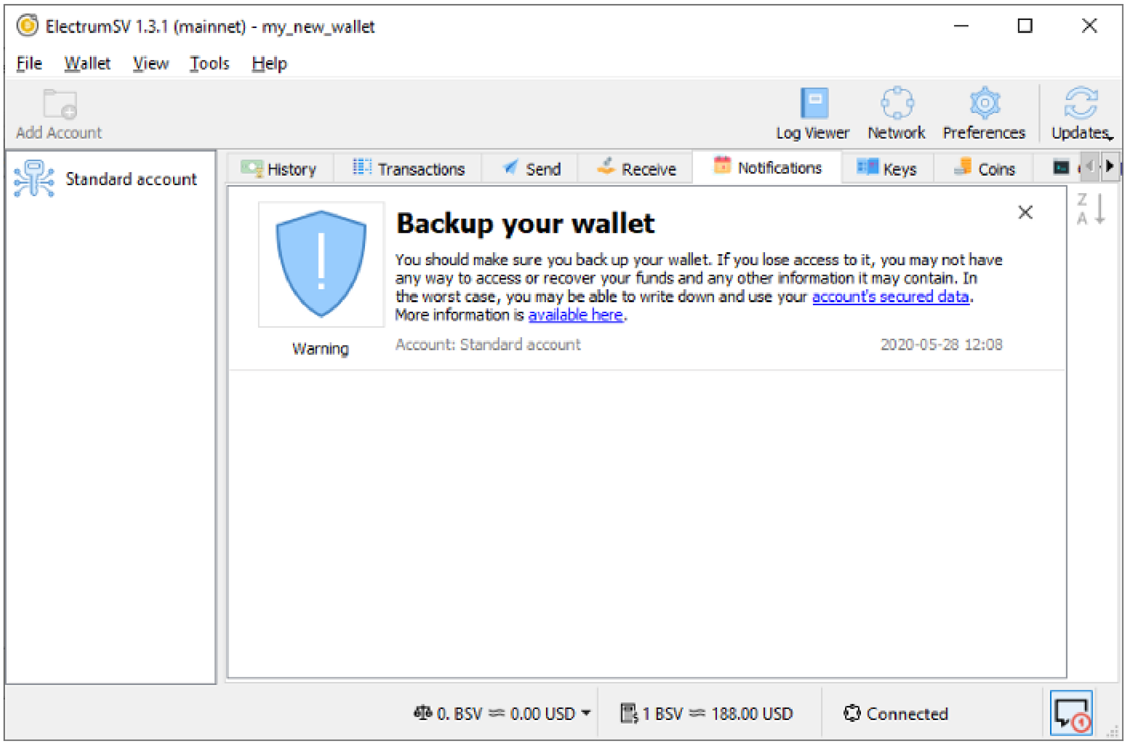
Clicking the notification icon will make the new "Notifications" tab the active one and show the initial notification about backing up your data.

1.3.5 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.



ElectrumSV 1.3.1 (mainnet) - my_new_wallet

Backup your wallet

You should make sure you back up your wallet. If you lose access to it, you may not have any way to access or recover your funds and any other information it may contain. In the worst case, you may be able to write down and use your account's secured data. More information is available here.

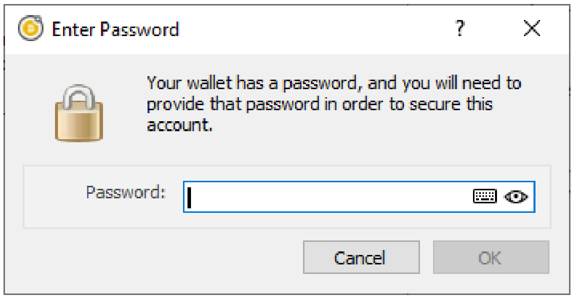
Account: Standard account 2020-05-28 12:08

0. BSV ≈ 0.00 USD

1 BSV ≈ 188.00 USD

Connected

Fig. 30: The initial backup notification in the wallet's notifications tab.

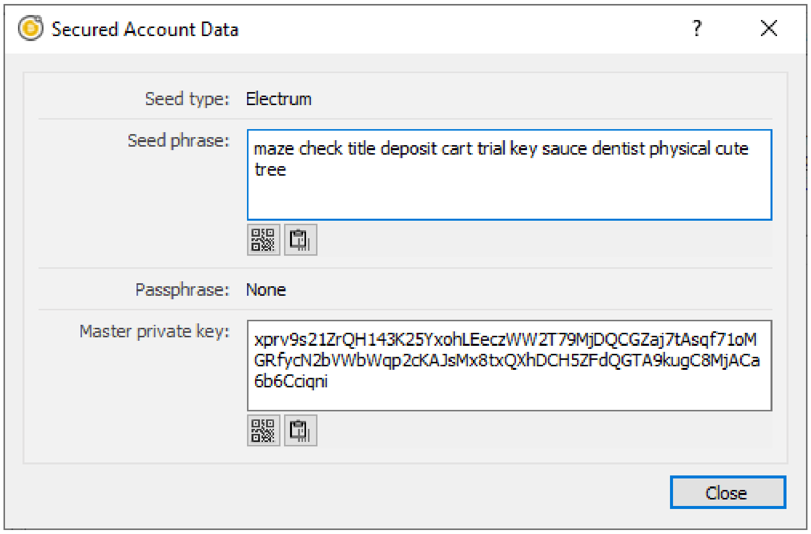


Enter Password

Your wallet has a password, and you will need to provide that password in order to secure this account.

Password:

Fig. 31: The password dialog.



Seed type: Electrum

Seed phrase: maze check title deposit cart trial key sauce dentist physical cute tree

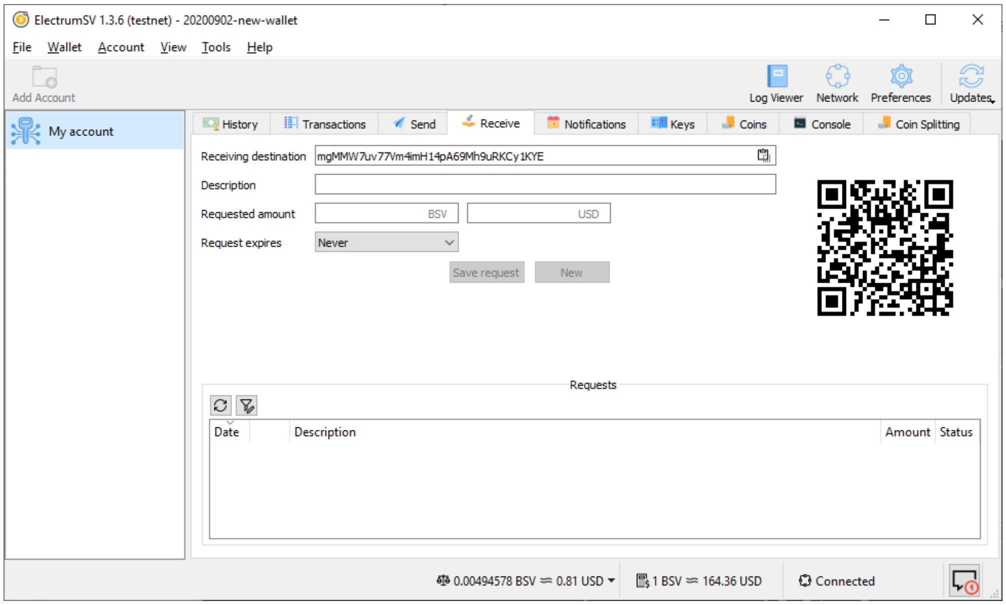
Passphrase: None

Master private key: xprv9s21ZrQH143K25YxohLEeczWW2T79MjDQCGZaj7tAsqf71oMGRfycN2bVWbWqp2cKAJsMx8txQXhDCH5ZFdQGTA9kugC8MjACa6b6Cciqni

Fig. 32: The secured data dialog.

1.4 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.



Receiving destination: mgMMW7uv77Vm4imH14pA69Mh9uRKCy1KYE

Description:

Requested amount: BSV / USD

Request expires: Never

0.00494578 BSV ≒ 0.81 USD

1 BSV ≒ 164.36 USD

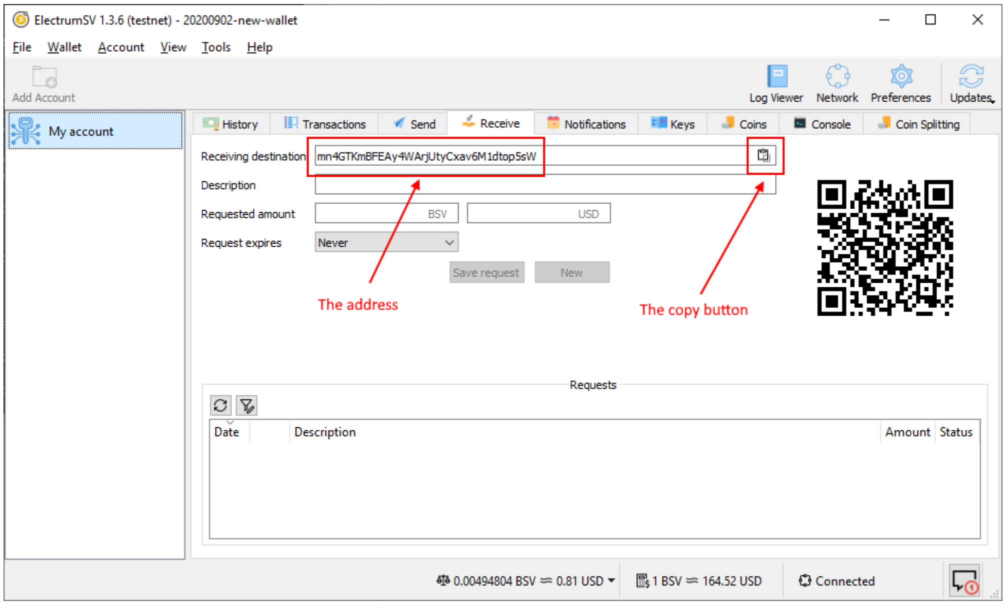
Fig. 33: The receiving tab in a standard account.

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.

1.4.1 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.



Receiving destination: mn4GTKmBFEAy4WArjUtyCxav6M1dtop5sW

Request expires: Never

0.00494804 BSV ≈ 0.81 USD

1 BSV ≈ 164.52 USD

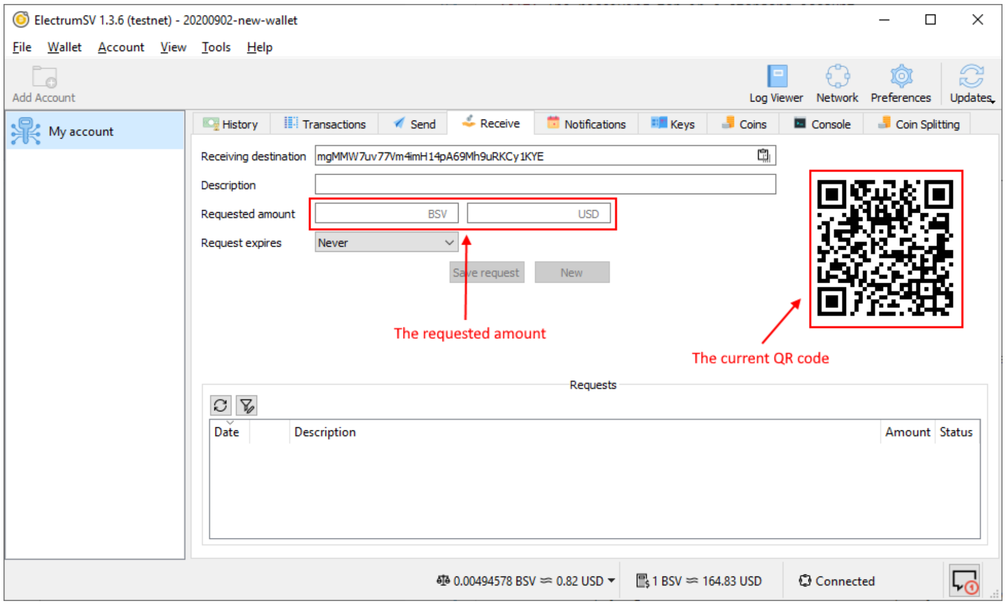
Fig. 34: The new address offered in the receiving tab.

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.

1.4.2 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.



Receiving destination: mgMMW7uv77Vm4imH14pA69Mh9uRKCy1KYE

Description:

Requested amount: BSV USD

Request expires: Never

0.00494578 BSV ≈ 0.82 USD

1 BSV ≈ 164.83 USD

Fig. 35: The QR code provided in the receiving tab.

1.4.3 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.

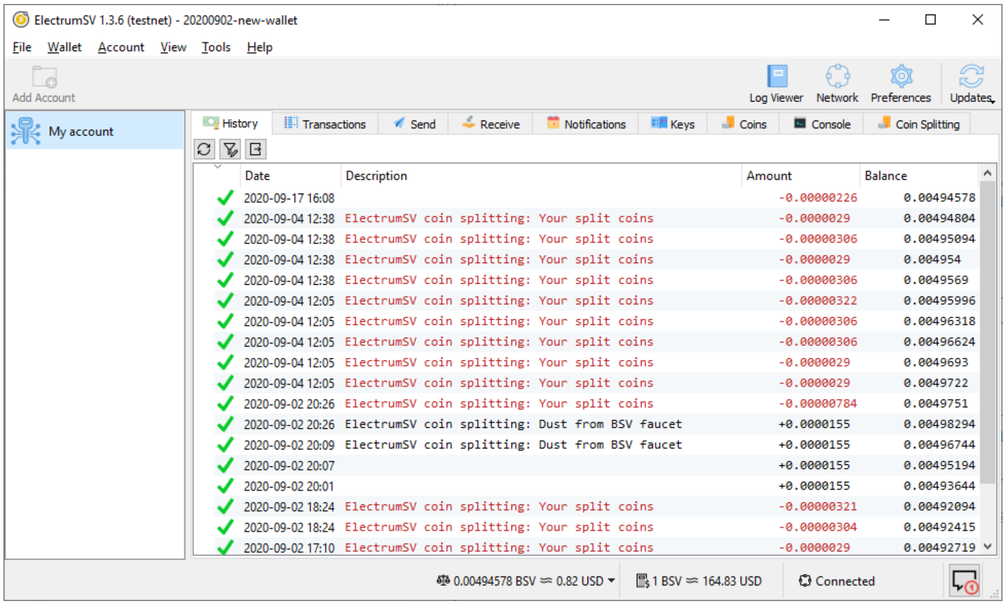
With this model, the wallet has no idea a payment is incoming until it arrives out of the blue. A new and better model is available in the form of Paymail, but ElectrumSV does not have the service infrastructure to support it at this time. We are however working towards it.

1.5 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.



	Date	Description	Amount	Balance
✓	2020-09-17 16:08		-0.00000226	0.00494578
✓	2020-09-04 12:38	ElectrumSV coin splitting: Your split coins	-0.0000029	0.00494804
✓	2020-09-04 12:38	ElectrumSV coin splitting: Your split coins	-0.00000306	0.00495094
✓	2020-09-04 12:38	ElectrumSV coin splitting: Your split coins	-0.0000029	0.004954
✓	2020-09-04 12:38	ElectrumSV coin splitting: Your split coins	-0.00000306	0.0049569
✓	2020-09-04 12:05	ElectrumSV coin splitting: Your split coins	-0.00000322	0.00495996
✓	2020-09-04 12:05	ElectrumSV coin splitting: Your split coins	-0.00000306	0.00496318
✓	2020-09-04 12:05	ElectrumSV coin splitting: Your split coins	-0.00000306	0.00496624
✓	2020-09-04 12:05	ElectrumSV coin splitting: Your split coins	-0.0000029	0.0049693
✓	2020-09-04 12:05	ElectrumSV coin splitting: Your split coins	-0.0000029	0.0049722
✓	2020-09-02 20:26	ElectrumSV coin splitting: Your split coins	-0.00000784	0.0049751
✓	2020-09-02 20:26	ElectrumSV coin splitting: Dust from BSV faucet	+0.0000155	0.00498294
✓	2020-09-02 20:09	ElectrumSV coin splitting: Dust from BSV faucet	+0.0000155	0.00496744
✓	2020-09-02 20:07		+0.0000155	0.00495194
✓	2020-09-02 20:01		+0.0000155	0.00493644
✓	2020-09-02 18:24	ElectrumSV coin splitting: Your split coins	-0.00000321	0.00492094
✓	2020-09-02 18:24	ElectrumSV coin splitting: Your split coins	-0.00000304	0.00492415
✓	2020-09-02 17:10	ElectrumSV coin splitting: Your split coins	-0.0000029	0.00492719

0.00494578 BSV ≈ 0.82 USD   1 BSV ≈ 164.83 USD   Connected

Fig. 36: The history tab when awaiting an incoming payment.

1.5.1 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.

1.5.2 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.

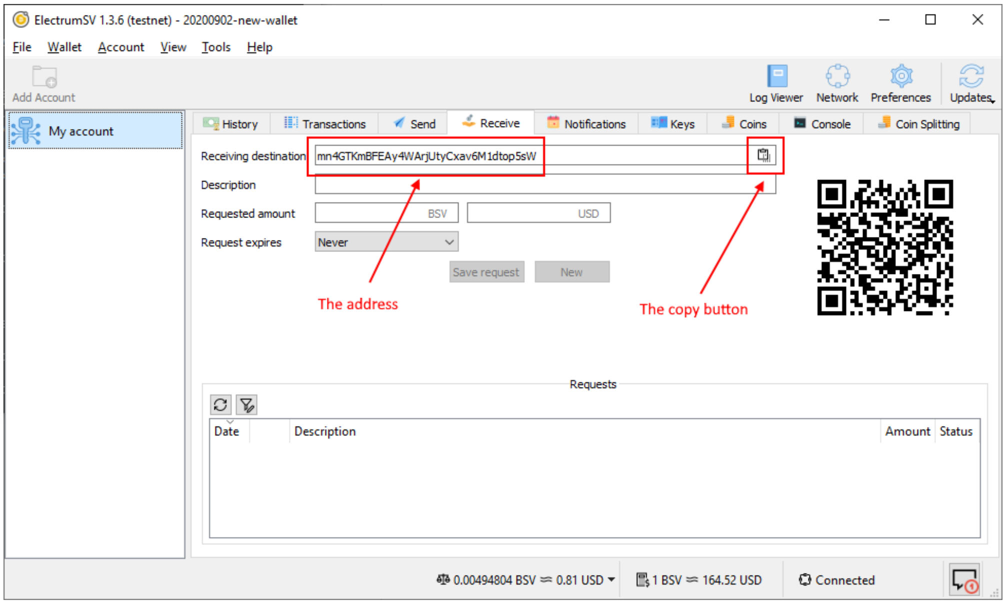
After pasting in the address, enter a nominal amount of Bitcoin SV to send, where your wallet has sufficient funds to do so.

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.



Receiving destination: mn4GTKmBFEAy4WArjUtyCxav6M1dtop5sW

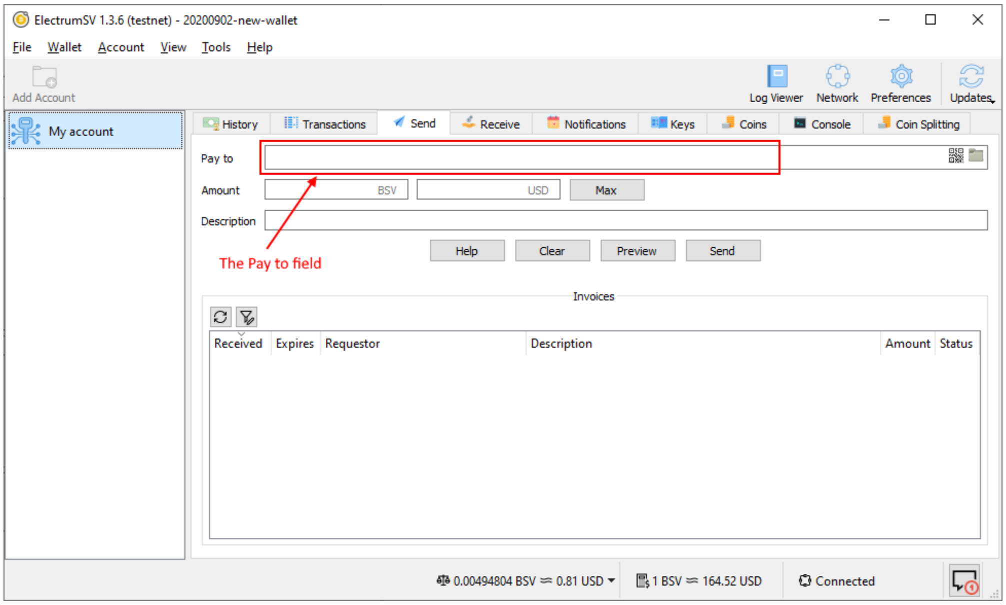
Request expires: Never

0.00494804 BSV ≈ 0.81 USD | 1 BSV ≈ 164.52 USD | Connected

The address

The copy button

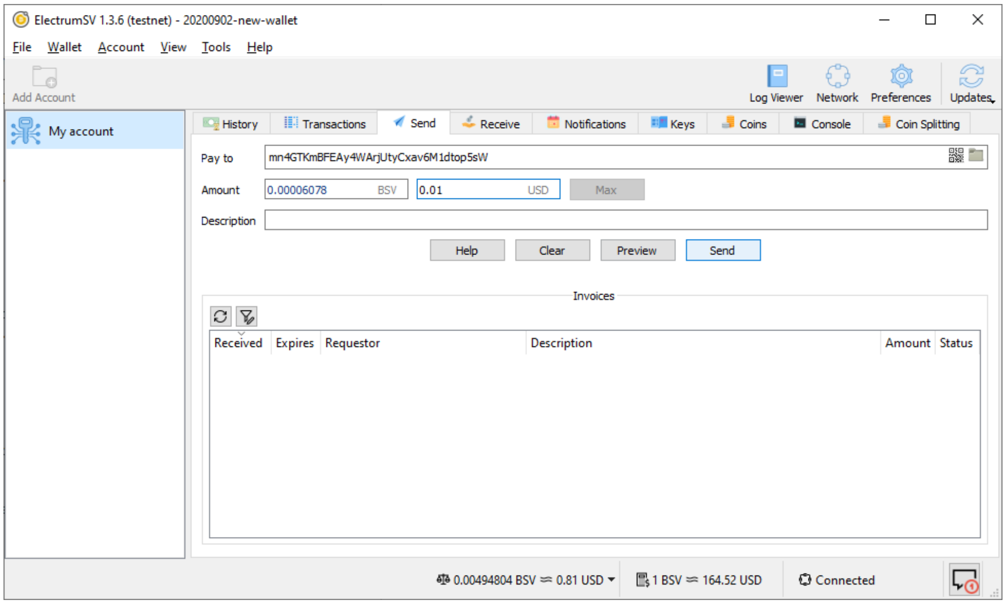
Fig. 37: Highlighted areas on the receiving tab.



0.00494804 BSV ≈ 0.81 USD | 1 BSV ≈ 164.52 USD | Connected

The Pay to field

Fig. 38: Highlighted areas on the send tab.



Pay to mn4GTKmBFEAy4WArjUtyCxav6M1dtop5sW

Amount 0.00006078 BSV 0.01 USD

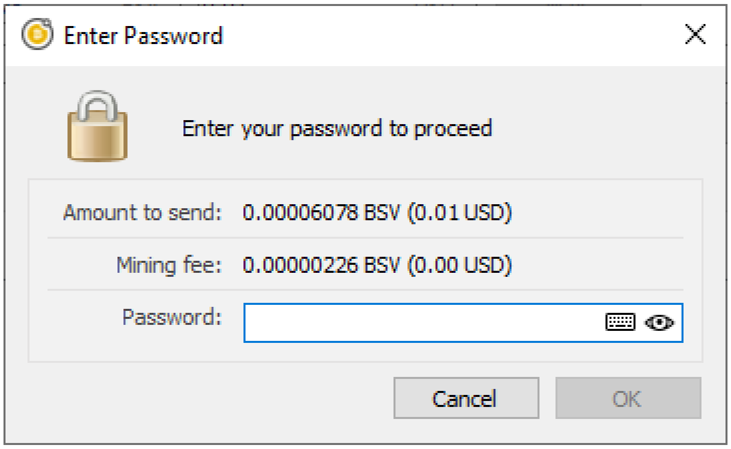
Description

0.00494804 BSV ≈ 0.81 USD

1 BSV ≈ 164.52 USD

Connected

Fig. 39: The filled out send tab.



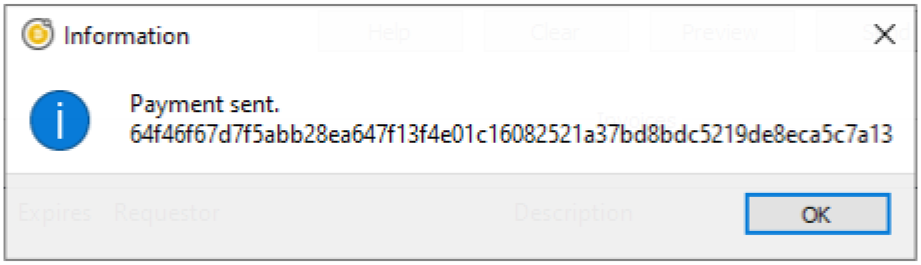
Enter your password to proceed

Amount to send: 0.00006078 BSV (0.01 USD)

Mining fee: 0.00000226 BSV (0.00 USD)

Password:

Fig. 40: The password confirmation dialog.

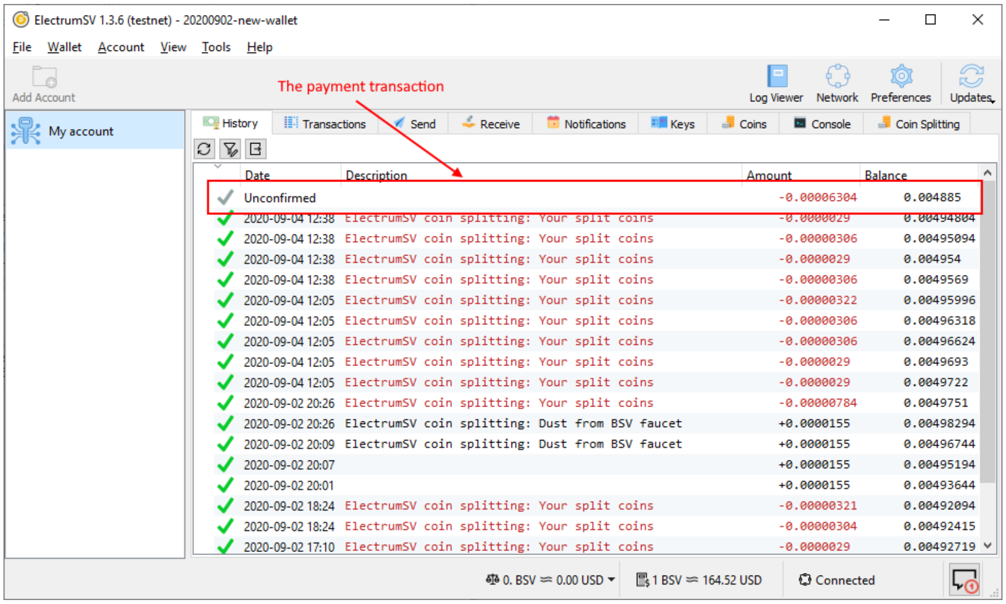


Payment sent.
64f46f67d7f5abb28ea647f13f4e01c16082521a37bd8bdc5219de8eca5c7a13

Fig. 41: The payment sent dialog.

1.5.3 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.



The payment transaction

Date	Description	Amount	Balance
✓ Unconfirmed		-0.00006304	0.004885
✓ 2020-09-04 12:38	ElectrumSV coin splitting: Your split coins	-0.0000029	0.00494804
✓ 2020-09-04 12:38	ElectrumSV coin splitting: Your split coins	-0.00000306	0.00495094
✓ 2020-09-04 12:38	ElectrumSV coin splitting: Your split coins	-0.0000029	0.004954
✓ 2020-09-04 12:38	ElectrumSV coin splitting: Your split coins	-0.00000306	0.0049569
✓ 2020-09-04 12:05	ElectrumSV coin splitting: Your split coins	-0.00000322	0.00495996
✓ 2020-09-04 12:05	ElectrumSV coin splitting: Your split coins	-0.00000306	0.00496318
✓ 2020-09-04 12:05	ElectrumSV coin splitting: Your split coins	-0.00000306	0.00496624
✓ 2020-09-04 12:05	ElectrumSV coin splitting: Your split coins	-0.0000029	0.0049693
✓ 2020-09-04 12:05	ElectrumSV coin splitting: Your split coins	-0.0000029	0.0049722
✓ 2020-09-02 20:26	ElectrumSV coin splitting: Your split coins	-0.00000784	0.0049751
✓ 2020-09-02 20:26	ElectrumSV coin splitting: Dust from BSV faucet	+0.0000155	0.00498294
✓ 2020-09-02 20:09	ElectrumSV coin splitting: Dust from BSV faucet	+0.0000155	0.00496744
✓ 2020-09-02 20:07		+0.0000155	0.00495194
✓ 2020-09-02 20:01		+0.0000155	0.00493644
✓ 2020-09-02 18:24	ElectrumSV coin splitting: Your split coins	-0.00000321	0.00492094
✓ 2020-09-02 18:24	ElectrumSV coin splitting: Your split coins	-0.00000304	0.00492415
✓ 2020-09-02 17:10	ElectrumSV coin splitting: Your split coins	-0.0000029	0.00492719

Fig. 42: 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*.

How do I split my coins?

If you have coins you have not touched since before Bitcoin SV and Bitcoin Cash split from each other, you might want to make sure that you can send one of these without accidentally sending the other. Read more about *coin splitting*.

2.1 Hardware wallet issues

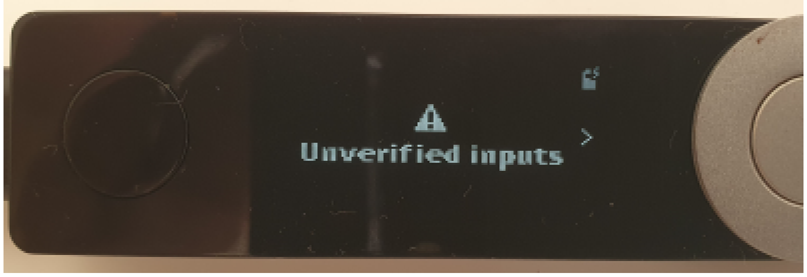
2.1.1 Ledger

While Ledger as a company do not support Bitcoin SV as a coin on their device, users have been able to use their Ledger devices with ElectrumSV through compatibility with the Bitcoin Cash support.

The Ledger device reports "unverified inputs"

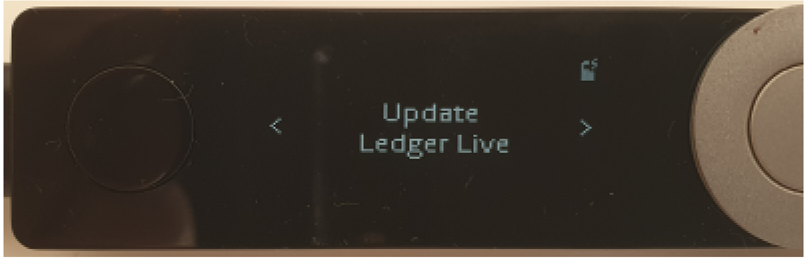
You go to sign your transaction and your Ledger device has a confusing series of screens talking about "unverified inputs" and updating your device and/or software. You can simply step through these screens and select continue. These screens will be shown below, and then a detailed explanation of why you are seeing them will be provided.

The short version is that you can continue past these screens to signing your transaction as you signed it before you started seeing these messages, and it will be as secure as it was then. Just make sure you only sign it once, and if ElectrumSV asks you to resign it over and over not recognising that you did it once, you are probably using malware. Again, see below the screens for an explanation of this in more detail.



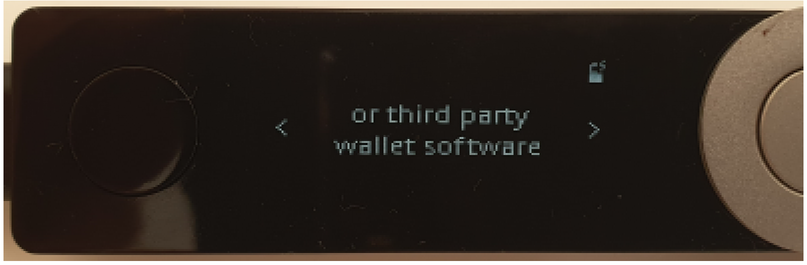
Unverified inputs

It should not be necessary to update your Ledger firmware and applications to deal with this.



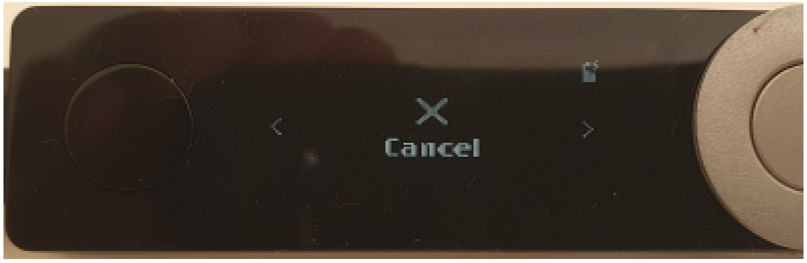
Update
Ledger Live

It should not be necessary to update ElectrumSV, although you should always be using the latest version.



or third party
wallet software

You can cancel the signing of the transaction if you want.



Cancel

But if you select the "continue" option, the Ledger device will go through the normal transaction signing process.

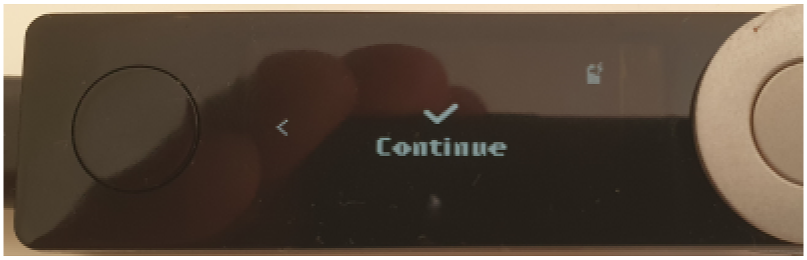
As you might recall, the first step of the correct signing process is to confirm where you are sending funds. And this is where the process is now at. You can go ahead and sign the transaction as you would have in the past before this confusing message.

Why do I see this "unverified inputs" message?

A theoretical but unlikely exploit was discovered where wallet malware could direct a user to sign a transaction several times, and extract the signed spends from each and combine them into a new transaction which gave a large fee to miners. Trezor wrote an article about it which you can read if you wish. You see this warning because ElectrumSV is not providing the previous transactions in which the spent coins were originally received to the Ledger device.

The simple reason we do not provide the previous transaction data is because Ledger cannot handle it and will break. You can see in the Trezor hardware issues a "DataError: bytes overflow" error, which their users may encounter. We have to provide these transactions to the Trezor devices but they cannot handle them and they break, this means that Trezor users have to be careful not to spend anything other than the simplest of received payments in their transactions themselves and work out what they can and can't spend themseves. If any of their coins is not simple and cannot be handled by Trezor, they need to bypass their hardware wallet and spend them in an unsafe way by entering their seed words.

Back to Ledger devices. Ledger allow the transaction to be signed without the spent transaction data, and on detecting they do not have it, they show an "unverified inputs" message. This makes it a little lot for Ledger users. They can still



sign a spend they are confident is going to the correct places, and not bypass their hardware wallet to do so. Let's be honest, if someone is going to all the effort of writing malware it has never in the history of malware been to give the stolen coins to miners. The chances of downloading malware are slight, and the chances of downloading malware that gives coins to anyone other than the thief are even slighter.

2.1.2 Trezor

While Trezor as a company do not support Bitcoin SV as a coin on their device, generally Bitcoin SV users have been able to use their Trezor devices with ElectrumSV by having it in the Bitcoin Cash coin mode. However, users are encountering situations where the limitations of the Trezor device result in it no longer being sufficient to work with Bitcoin SV transactions. This likely means that if a user is planning to continue to use a Trezor device, it may require them to jump through hoops to do so.

There are two complications:

- Later versions of firmware (starting with 1.9.1 for One and 2.3.1 for Model T) require ElectrumSV to pass in parent transactions with the transaction you are signing. ElectrumSV only started supporting this in ElectrumSV 1.3.8 or newer. What this means is that if you are using these later versions of firmware, you must be using ElectrumSV 1.3.8 or newer - or it will error.
- Bitcoin SV transactions can have large output scripts, larger than what Trezor can handle. Trezor can only sign simple payments and nothing else, but this does not prevent payments from being made into the wallet with additional output scripts added for other reasons that exceed Trezor's size limit of 15 kilobytes. The parent transaction processing in the Trezor device will error when it encounters these.

Trezor devices are becoming problematic for Bitcoin SV users to use. While they are polished and enjoyable devices to use, unless the large output problem is solved by Trezor, we cannot recommend users buy these devices unless they accept they have to own and deal with these problems. For this reason it is recommended that Trezor users downgrade their devices.

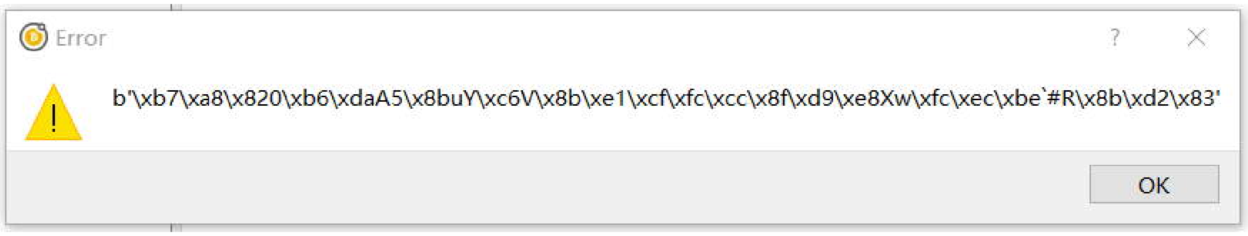
Downgrading your Trezor device

These are Trezor's firmware version pages, for users who plan to downgrade:

- Trezor One: 1.9.0.
- Trezor Model T: 2.3.0.

You will need to visit those pages and download the firmware file. Trezor provide instructions on how to downgrade, and let you know how and where to use the file.

Problem: You see a random looking series of numbers and letters

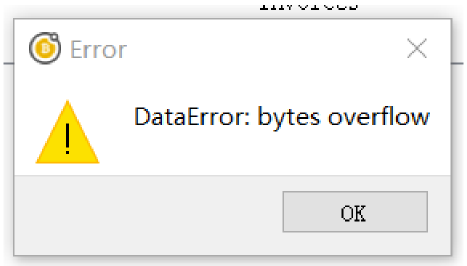


```
b'\xb7\xa8\x820\xb6\xdaA5\x8buY\xc6V\x8b\xe1\xcf\xfc\xcc\x8f\xd9\xe8Xw\xfc\xec\xbe`#R\x8b\xd2\x83'
```

Fig. 1: What this problem looks like..

You are using ElectrumSV 1.3.7 or earlier, and your Trezor device has a later version of the firmware. It expects ElectrumSV to have provided the transaction associated with those numbers and letters, but the ElectrumSV version you are using does not know how to or even that it should. You can take the risk of updating to a more recent version of ElectrumSV that supports these parent transactions, and possibly encounter the "DataError: bytes overflow" problem. Or you can downgrade your Trezor firmware to the version listed above.

Problem: You see the message "DataError: bytes overflow"



```
DataError: bytes overflow
```

Fig. 2: What this problem looks like..

One of your parent transactions contains not only the coin you are trying to spend, but a large output script. Your Trezor device has a later version of firmware where parent transactions are required to be provided, and the device is choking on the large output. This is a limit in the device itself, and ElectrumSV can do nothing about this. To spend the coin associated with the problem parent transaction, you need to downgrade your firmware to the versions listed above.

2.2 Coin splitting

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

When users have coins that existed before Bitcoin Cash became a separate blockchain from Bitcoin SV, those coins are linked on both blockchains. When they are sent in a wallet on one blockchain, that action can also send them on the other blockchain. Users have had this accidentally happen to them, and the recipient has refused to refund the coins from the blockchain the user did not intend to send on.

If you think you have unsplit coins in your wallet, you can use ElectrumSV's coin-splitting feature to split them. But keep in mind that you are responsible for your own coins, you should verify for yourself that the splitting worked. And if you are unsure whether your coins need to be split, you can always split them anyway.

2.2.1 How does splitting work?

The process is simple, if the coins are sent on Bitcoin SV in a way that is incompatible with Bitcoin Cash, then the coins are split. Any usage of those specific coins that have been split will from then on be independent on either blockchain.

In order to keep it simple, we only do the simplest case. We make your wallet do a payment to itself that combines all the available coins within it in a way that should be valid on Bitcoin SV and not Bitcoin Cash. This results in one single split coin combining all the individual coins that you had in your wallet before the split.

2.2.2 How you split your coins

Unfortunately, all the coins in the wallet used here are already split. So the following is just going through the process to show you how it works. You can see that this wallet contains a small amount of Bitcoin SV.

Let's start by changing to the coin-splitting tab:

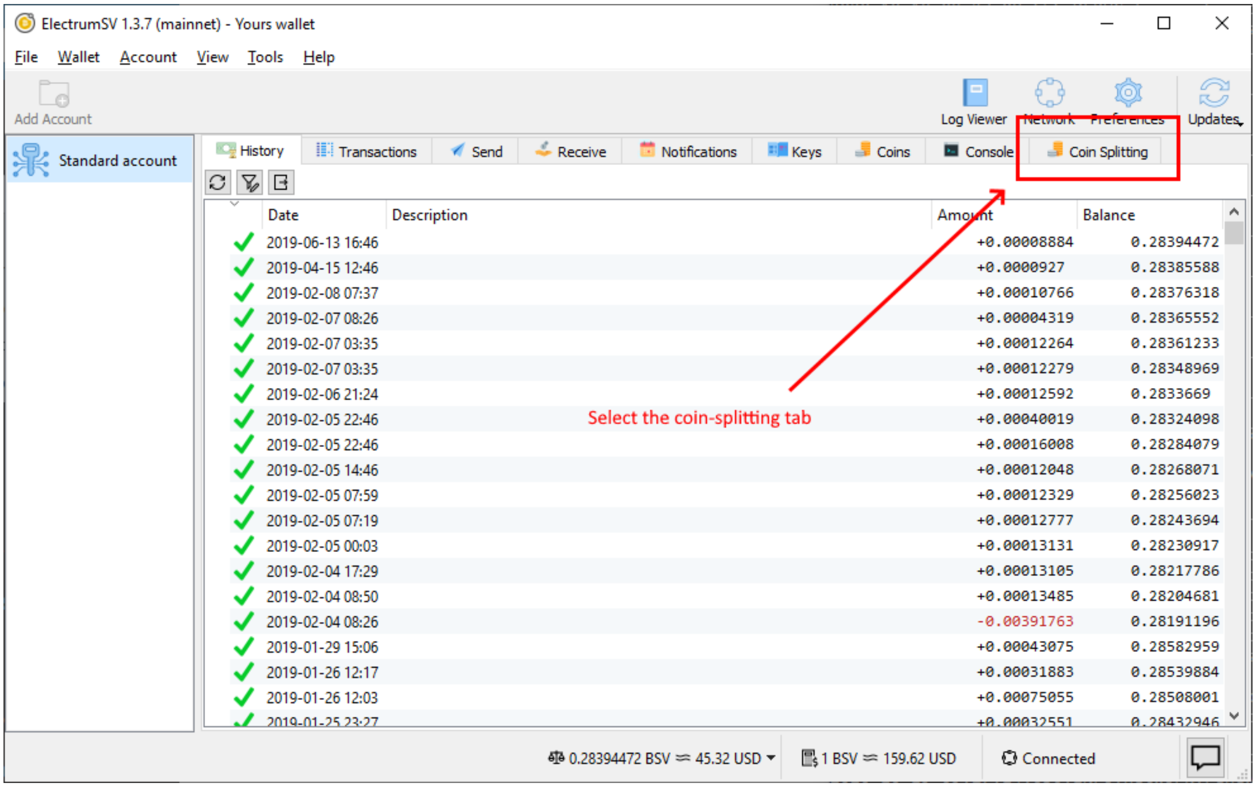
Once you are looking at the coin-splitting tab, you have two options. Either direct splitting or faucet splitting. We recommend the direct splitting, and do not really support the faucet splitting any more. Direct splitting does not work for hardware wallets, which due to inherent limitations can only work in simple ways.

Clicking on the direct splitting button will ask you for your password. You will see that the balance of the splitting transaction is the balance of the available coins in the wallet.

After you enter your password, it will sign and broadcast your transaction. This will happen pretty quickly, and once it is done you will see a dialog letting you know the splitting transaction was broadcast.

You can now go back to the history tab and see the splitting transaction there, which has an automatic description noting what it was created for.

In theory, your coins should be split. But again, you are responsible for using them safely and you should ensure that they are really split.

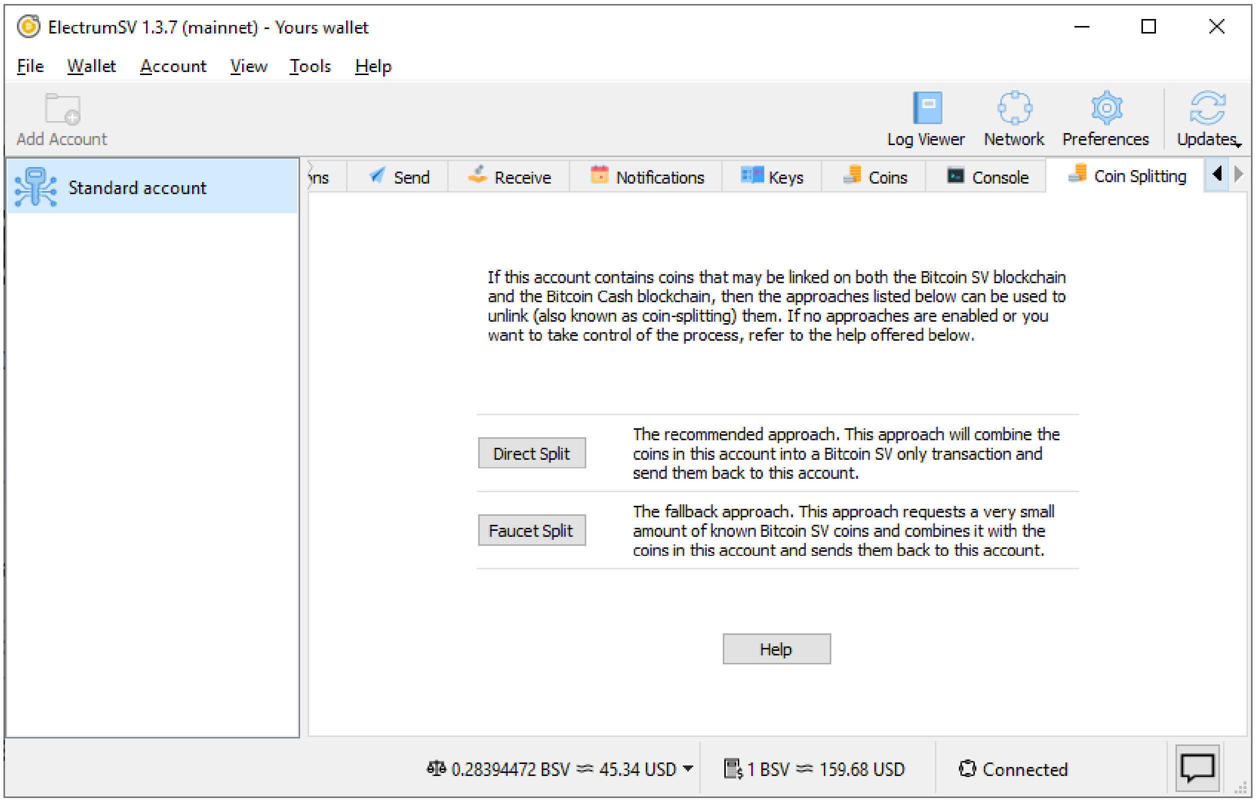


	Date	Description	Amount	Balance
✓	2019-06-13 16:46		+0.00008884	0.28394472
✓	2019-04-15 12:46		+0.0000927	0.28385588
✓	2019-02-08 07:37		+0.00010766	0.28376318
✓	2019-02-07 08:26		+0.00004319	0.28365552
✓	2019-02-07 03:35		+0.00012264	0.28361233
✓	2019-02-07 03:35		+0.00012279	0.28348969
✓	2019-02-06 21:24		+0.00012592	0.2833669
✓	2019-02-05 22:46		+0.00040019	0.28324098
✓	2019-02-05 22:46		+0.00016008	0.28284079
✓	2019-02-05 14:46		+0.00012048	0.28268071
✓	2019-02-05 07:59		+0.00012329	0.28256023
✓	2019-02-05 07:19		+0.00012777	0.28243694
✓	2019-02-05 00:03		+0.00013131	0.28230917
✓	2019-02-04 17:29		+0.00013105	0.28217786
✓	2019-02-04 08:50		+0.00013485	0.28204681
✓	2019-02-04 08:26		-0.00391763	0.28191196
✓	2019-01-29 15:06		+0.00043075	0.28582959
✓	2019-01-26 12:17		+0.00031883	0.28539884
✓	2019-01-26 12:03		+0.00075055	0.28508001
✓	2019-01-25 23:27		+0.00032551	0.28432946

Select the coin-splitting tab

0.28394472 BSV ≈ 45.32 USD   1 BSV ≈ 159.62 USD   Connected

Fig. 3: Selecting the coin-splitting tab.



If this account contains coins that may be linked on both the Bitcoin SV blockchain and the Bitcoin Cash blockchain, then the approaches listed below can be used to unlink (also known as coin-splitting) them. If no approaches are enabled or you want to take control of the process, refer to the help offered below.

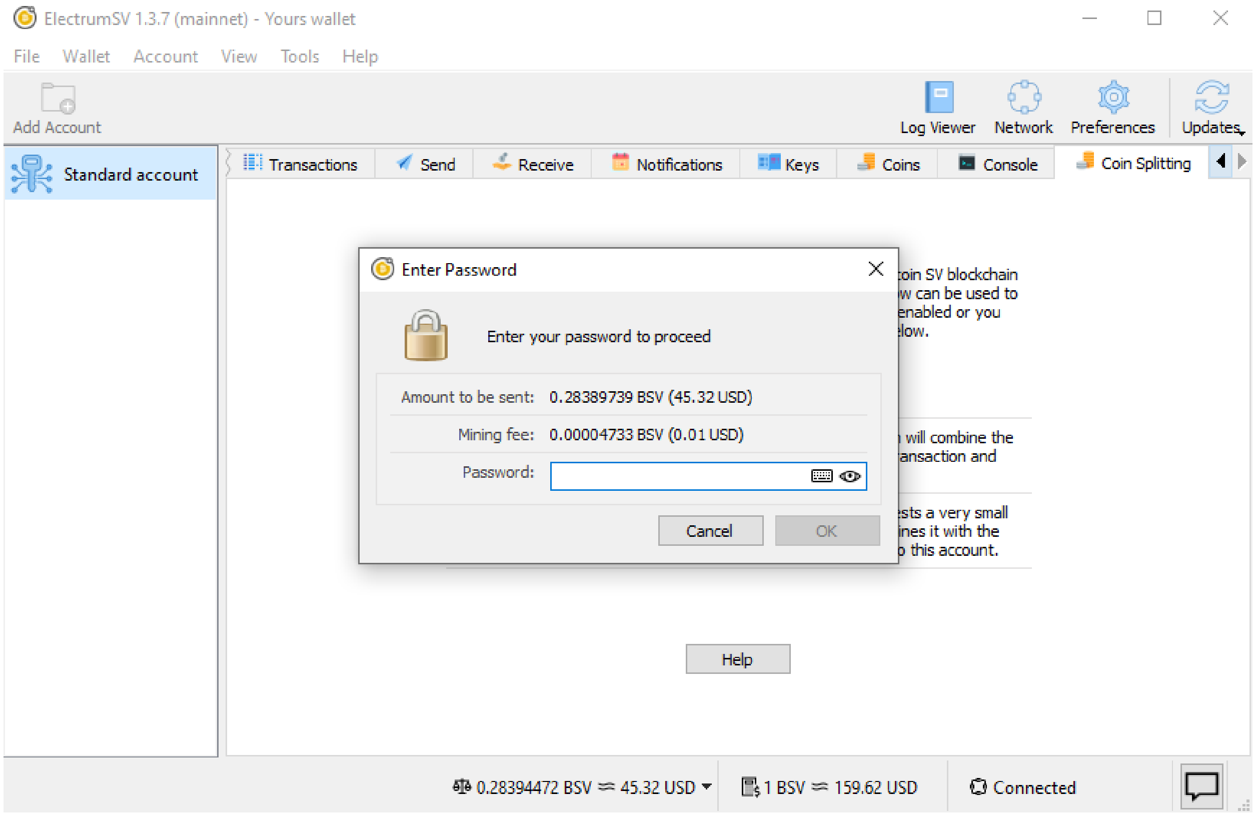
Direct Split — The recommended approach. This approach will combine the coins in this account into a Bitcoin SV only transaction and send them back to this account.

Faucet Split — The fallback approach. This approach requests a very small amount of known Bitcoin SV coins and combines it with the coins in this account and sends them back to this account.

Help

0.28394472 BSV ≈ 45.34 USD   1 BSV ≈ 159.68 USD   Connected

Fig. 4: The coin-splitting tab.



Enter your password to proceed

Amount to be sent: 0.28389739 BSV (45.32 USD)

Mining fee: 0.00004733 BSV (0.01 USD)

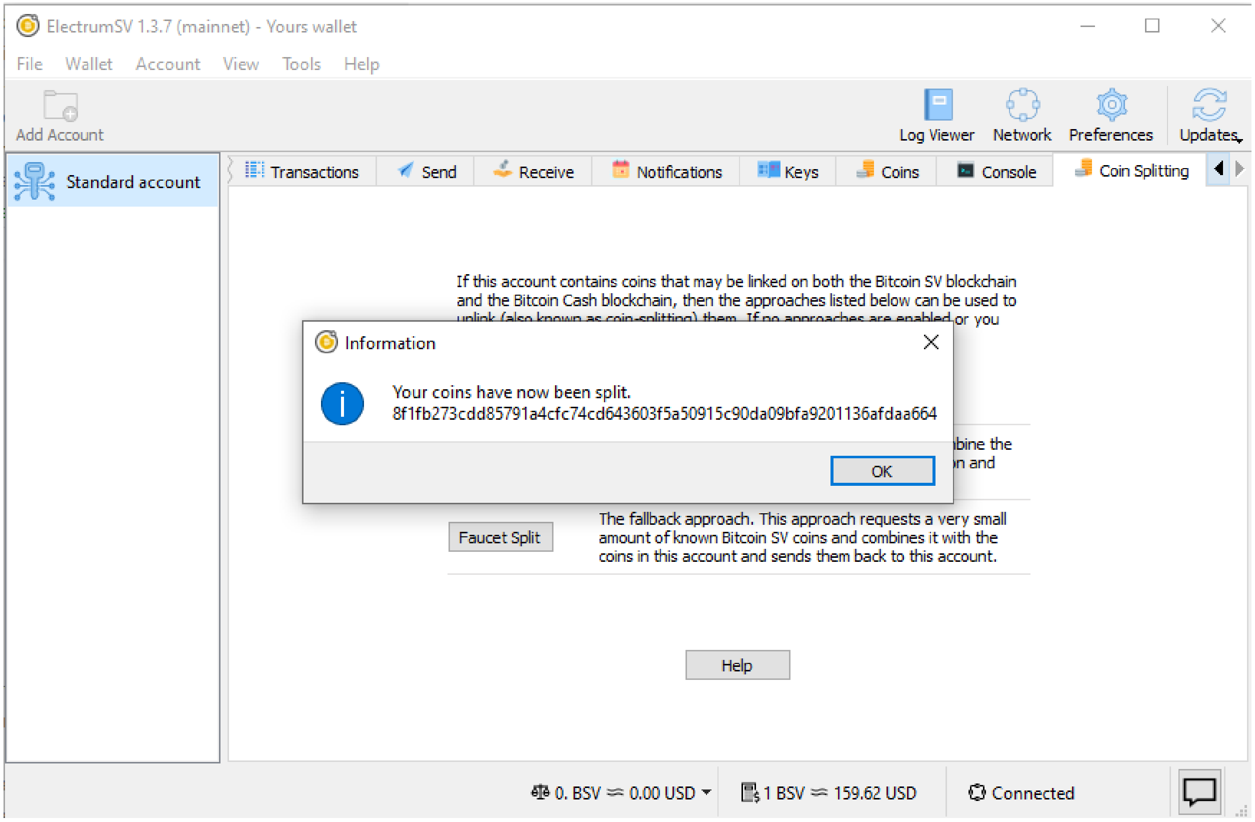
Password:

0.28394472 BSV ≈ 45.32 USD

1 BSV ≈ 159.62 USD

Connected

Fig. 5: Approve the splitting payment.



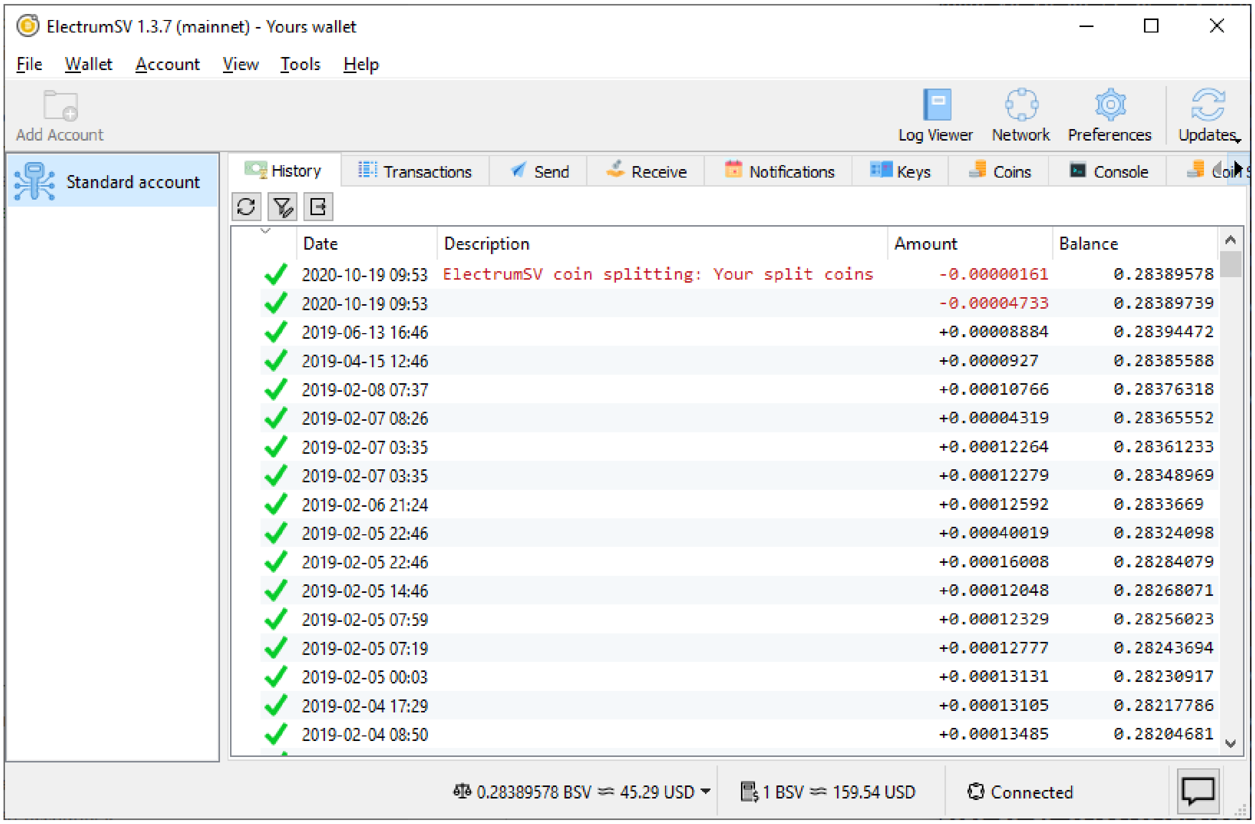
If this account contains coins that may be linked on both the Bitcoin SV blockchain and the Bitcoin Cash blockchain, then the approaches listed below can be used to unlink (also known as coin-splitting) them. If no approaches are enabled or you

Your coins have now been split.
8f1fb273cdd85791a4cfc74cd643603f5a50915c90da09bfa9201136afdaa664

bine the
n and

The fallback approach. This approach requests a very small amount of known Bitcoin SV coins and combines it with the coins in this account and sends them back to this account.

Fig. 6: The split action completion message.



	Date	Description	Amount	Balance
✓	2020-10-19 09:53	ElectrumSV coin splitting: Your split coins	-0.00000161	0.28389578
✓	2020-10-19 09:53		-0.00004733	0.28389739
✓	2019-06-13 16:46		+0.00008884	0.28394472
✓	2019-04-15 12:46		+0.0000927	0.28385588
✓	2019-02-08 07:37		+0.00010766	0.28376318
✓	2019-02-07 08:26		+0.00004319	0.28365552
✓	2019-02-07 03:35		+0.00012264	0.28361233
✓	2019-02-07 03:35		+0.00012279	0.28348969
✓	2019-02-06 21:24		+0.00012592	0.2833669
✓	2019-02-05 22:46		+0.00040019	0.28324098
✓	2019-02-05 22:46		+0.00016008	0.28284079
✓	2019-02-05 14:46		+0.00012048	0.28268071
✓	2019-02-05 07:59		+0.00012329	0.28256023
✓	2019-02-05 07:19		+0.00012777	0.28243694
✓	2019-02-05 00:03		+0.00013131	0.28230917
✓	2019-02-04 17:29		+0.00013105	0.28217786
✓	2019-02-04 08:50		+0.00013485	0.28204681

0.28389578 BSV ≂ 45.29 USD   1 BSV ≂ 159.54 USD   Connected

Fig. 7: The history tab with the splitting transaction.

2.2.3 Ensuring your coins are split

Bitcoin is complicated, and in order to really know for yourself that your coins are split, you need to have some level of technical understanding. It's a lot simpler to just send them to different places on both blockchains, especially safe places like your own wallet's receiving addresses and check that they get there - so just do that!

Here is one way to do it:

1. Do a direct split in ElectrumSV.
2. Open your Bitcoin Cash wallet with the coins that were linked to Bitcoin SV, that you just split in ElectrumSV.
3. Create a new empty Bitcoin Cash wallet.
4. Send the coins in your existing Bitcoin Cash wallet to the new Bitcoin Cash wallet.

You can then observe that your Bitcoin Cash is in a new fresh wallet, and your Bitcoin SV is in the old wallet. Neither moved because the other moved, but rather both were moved by you. You might wonder why you need to create a second Bitcoin Cash wallet, and the reason is that this ensures that your Bitcoin SV and Bitcoin Cash are using different keys and it both helps verify they are unlinked and gives you better security going forward.

2.2.4 Hardware wallets

Hardware wallets are extremely limited devices with not much flexibility. They only allow certain types of transactions to be signed, and this does not include the type that the direct splitting method uses.

If you have a hardware wallet, you can try and use faucet splitting. Faucet splitting works by adding a very small Bitcoin SV coin to your wallet, then combining all the available coins in your wallet with that Bitcoin SV coin. This creates a new Bitcoin SV coin which is of course incompatible with the Bitcoin Cash blockchain, and so the coins in the wallet have been split.

Alternatively, if the faucet is not working you can get someone to send you a very small amount of Bitcoin SV and you can accomplish the same thing yourself by sending all the coins in your wallet to one of your own addresses (including that very small amount of Bitcoin SV).

2.2.5 Increasing differences between blockchains

There are an increasing number of changes between Bitcoin Cash and Bitcoin SV. While it is good practice to split your coins just in case you lose your Bitcoin SV when sending your Bitcoin Cash, or lose your Bitcoin Cash when sending your Bitcoin SV, it is possibly becoming easier to avoid it.

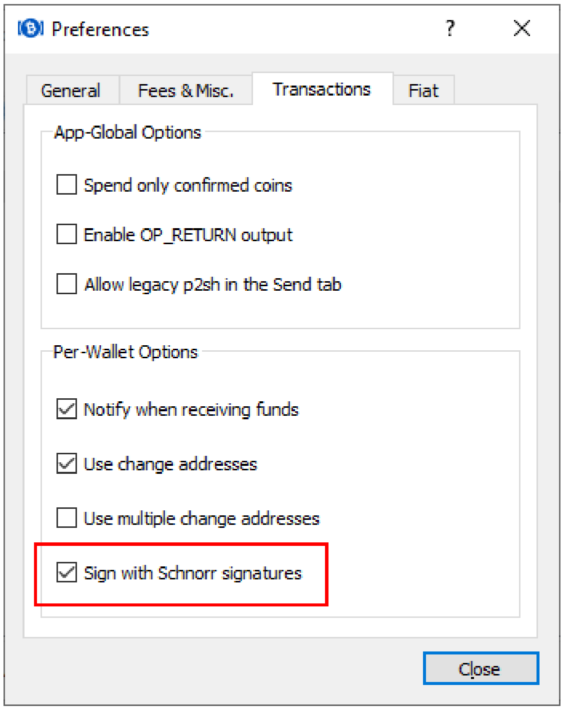
High minimum fee on Bitcoin Cash

The Bitcoin Cash servers for the Electron Cash wallet rejected any attempt to broadcast a transaction containing unsplit coins that had 0.5 satoshis per byte fee as too low. Experiments suggest that it is very difficult to get a transaction at this fee level to propagate, maybe nearing impossible.

As the default fee in ElectrumSV is 0.5 satoshis per byte, this could mean that if you send unsplit coins in ElectrumSV the Bitcoin Cash network will completely ignore them. Should you rely on this? No, but it might provide a coincidental safety net for people who do not know they should split their coins.

Schnorr signatures

By default Electron Cash and likely all Bitcoin Cash wallets now use Schnorr signatures. What this means is that the transactions they make should be incompatible with Bitcoin SV as long as the user has not opted out of using Schnorr. So in theory you can just send your coins on Bitcoin Cash and because those Schnorr signatures are used, the coins on Bitcoin Cash have been sent in a way that is incompatible with Bitcoin SV.



App-Global Options

☐ Spend only confirmed coins

☐ Enable OP_RETURN output

☐ Allow legacy p2sh in the Send tab

Per-Wallet Options

☑ Notify when receiving funds

☑ Use change addresses

☐ Use multiple change addresses

☑ Sign with Schnorr signatures

Fig. 8: The default Electron Cash Schnorr setting.

Should you rely on this? Not unless you know for sure that you are using Schnorr signatures in your Bitcoin Cash wallet, and that you have used the correctly.

2.2.6 Thanks

Many thanks to satoshi.io who provided unsplit coins used for testing related to this article.

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`

Sample Response

```
{
    "wallets": [
        "worker1.sqlite"
    ]
}
```

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`

Sample Response

```
{
    "parent_wallet": "worker1.sqlite",
    "accounts": {
        "1": {
            "wallet_type": "Standard account",
            "default_script_type": "P2PKH",
            "is_wallet_ready": true
        }
    }
}
```

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`

Sample Response

```
{
    "parent_wallet": "worker1.sqlite",
    "accounts": {
        "1": {
            "wallet_type": "Standard account",
            "default_script_type": "P2PKH",
            "is_wallet_ready": true
        }
    }
}
```

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`

Sample Response

```
{
    "1": {
        "wallet_type": "Standard account",
        "default_script_type": "P2PKH",
        "is_wallet_ready": true
    }
}
```

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`

Sample Response

```
{
    "cleared_coins": 11,
    "settled_coins": 700,
    "unmatured_coins": 0
}
```

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`

Sample Response

```
{
    "utxos": [
        {
            "value": 20000,
            "script_pubkey": "76a91485324d225c81d414fe8a92bf101dba1a59211e8488ac",
            "script_type": 2,
            "tx_hash": "ce7c2fbc25d25d945b4ad539d2b41ead29e1b786a8aa42b2677af28da3f231a0
↪",
            "out_index": 49,
            "keyinstance_id": 13,
            "address": "msfERZdhGaabQmeQ1ks8sHYdCDtxnTfL2z",
            "is_coinbase": false,
            "flags": 0
        },
        {
            "value": 20000,
            "script_pubkey": "76a91488471d45666dadece7f06aca22f1a1cf9a3a534988ac",
            "script_type": 2,
            "tx_hash": "ce7c2fbc25d25d945b4ad539d2b41ead29e1b786a8aa42b2677af28da3f231a0
↪",
            "out_index": 50,
            "keyinstance_id": 12,
            "address": "mswXPFgWJbgvyxkWBFfYjbbaD1DZmFS3ig",
            "is_coinbase": false,
            "flags": 0
        },
    ]
}
```

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`

Sample Response

```
{
    "confirmed_balance": 14999694400,
    "unconfirmed_balance": 98000,
    "unmatured_balance": 0
}
```

remove

Removes transactions (currently restricted to ‘StateSigned’ transactions.)

Deleting transactions in the ‘Dispatched’, ‘Cleared’, ‘Settled’ states could cause issues with the utxo set 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`

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

Sample Body Payload

```
{
    "txids": [
        "96eee07f8e2c96e33d457138496958d912042ff4ed7b3b9c74a2b810fa5c3750",
        "469ddc27b8ef3b386bf7451aebce64edfe22d836ad51076c7a82d78f8b4f4cf9",
        "e81472f9bbf2dc2c7dcc64c1f84b91b6214599d9c79e63be96dcda74dcb8103d"
    ]
}
```

Sample Response

```
{
    "items": [
        {
            "id": "96eee07f8e2c96e33d457138496958d912042ff4ed7b3b9c74a2b810fa5c3750",
            "result": 200
        },
        {
            "id": "469ddc27b8ef3b386bf7451aebce64edfe22d836ad51076c7a82d78f8b4f4cf9",
            "result": 400,
            "description": "DisabledFeatureError: You used this endpoint in a way that␣
↪is not supported for safety reasons. See documentation for details (https://electrumsv.
↪readthedocs.io/ )"
        },
        {
            "id": "e81472f9bbf2dc2c7dcc64c1f84b91b6214599d9c79e63be96dcda74dcb8103d",
            "result": 400,
            "description": "Transaction not found"
        }
    ]
}
```

get_transaction_history

Get transaction history. `tx_flags` can be specified in the request body. This is an enum representing a bitmask for filtering transactions.

The main `TxFlags` are:

StateCleared
1 << 20 (received over p2p network and is unconfirmed and in the mempool)

StateSettled
1 << 21 (received over the p2p network and is confirmed in a block)

StateReceived
1 << 22 (received from another party and is unknown to the p2p network)

StateSigned
1 << 23 (not sent or given to anyone else, but are with-holding and consider the inputs it uses allocated)

StateDispatched
1 << 24 (a transaction you have given to someone else, and are considering the inputs it uses allocated)

However, there are other flags that can be set. See `electrumsv/constants.py:TxFlags` for details.

In the example below, (1 << 23 | 1 << 21) yields 9437184 (to filter for only StateSigned and StateCleared transactions)

An empty request body will return all transaction history for this account. Pagination is not yet implemented.

Request

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 Body Payload

```
{
    "tx_flags": 9437184
}
```

Sample Response

```
{
    "history": [
        {
            "txid": "64a9564588f9ebcce4ac52f4e0c8fe758b16dfd6fdb5bd8db5920da317aa15c8",
            "height": 0,
            "tx_flags": 1052720,
            "value": -10200
        },
        {
            "txid": "a6ec24243a79de1b51646d1a46ece854a8f682ff23b4d4afabaebc2bc10ef110",
            "height": 0,
            "tx_flags": 1052720,
            "value": -10200
        }
    ]
}
```

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

```
{
    "tx_hex":
↪"0100000001e59dd2992ed46911bea87af1b4f7ab1edce8e038520f142d2aa219492664d993160000006b483045022100ec97
↪"
}
```

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 World" encoded in Hex. The preceeding 0x0b byte represents a pushdata op code to push the next 11 bytes onto the stack ("68656c6c6f20776f726c64").

Additional outputs for leftover change will be created automatically.

```
{
    "outputs": [
        {"script_pubkey":"006a0b68656c6c6f20776f726c64", "value": 0}
    ],
    "password": "test"
}
```

Sample Response

```
{
    "txid": "96eee07f8e2c96e33d457138496958d912042ff4ed7b3b9c74a2b810fa5c3750",
    "rawtx":
↪"0100000001cfdec4ce0f10c4148b44163bf6205f53e5ab31f04a57fcaaeb33ef6487e08511000000006b483045022100873b
↪"
}
```

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 World" encoded in Hex. The preceeding 0x0b byte represents a pushdata op code to push the next 11 bytes onto the stack ("68656c6c6f20776f726c64").

Additional outputs for leftover change will be created automatically.

```
{
    "rawtx":
↪"0100000001ab9aff89a92c011b5436a0c02eb53cf6328286e5cf5767f309cde5414f6576610000000006a473044022050750e
↪"
}
```

Sample Response

```
{
    "txid": "7ff0fcf6de91ffa71ef145e31d0bffe31467ecaa125a8db307cf9066fea55db5"
}
```

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 World" encoded in Hex. The preceeding 0x0b byte represents a pushdata op code to push the next 11 bytes onto the stack ("68656c6c6f20776f726c64").

Additional outputs for leftover change will be created automatically.

```
{
    "outputs": [
        {"script_pubkey":"006a0b68656c6c6f20776f726c64", "value": 0}
    ],
    "password": "test"
}
```

Sample Response

```
{
    "txid": "469ddc27b8ef3b386bf7451aebce64edfe22d836ad51076c7a82d78f8b4f4cf9"
}
```

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

```
{
    "txid": "42329848db94cb16379b0c8898eb2b98542fb25d9257a47663c3fac7b0f49938"
}
```

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

```
{
    "txid": "8f3dfe9b9e84c1d0b6d6ead8700be4114bb2d3ca1f97e1e84c64ea944415c723"
}
```

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

```
{
    "txid": [
        "72d1270d0b3ad4c71d8257db8d6f880186108152534658ae6a127b616795530d"
    ]
}
```

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

```
{
    "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

There are a lot of steps to releasing a new version of ElectrumSV. This document is intended to lay out the entire proces and some of the reasoning behind it, so that any developer can jump in and do a release if necessary. In addition, formalising the release process ensures that nothing is accidentally left out due to any informal and casually documented process leading to an oversight of various steps.

4.3.1 Initial preparation

When it is time to release a new version, the first step is to freeze the release branch in Github and prevent introduction of any changes that introduce new functionality or change existing functionality. This is an exercise in self restraint, rather than anything that is done to programmatically disallow these changes to be made.

Writing an article

An initial outline of a release article is written, including the featured changes that will be highlighted. Mostly this involves taking the last article, removing all the changes that were included in the previous version, and putting the new version's changes in their place using the same format. The key goal of these articles is to illustrate these changes and help users visualise them even if they skim through the article, and it should include screenshots at every possible opportunity.

For each change featured in a release article:

- A link should be provided to any issue that exists in relation to that change.
- A link should be provided to every code change made to the source code in the making of the given change.

Updating the version

The version number is increased to the new version number, and the approximate release date is updated to be approximately what it will be when the release is made. If the release process is protracted over many days due to the testing, and any subsequent changes they require, then the date may be modified later.

If the last version was `1.3.6`:

- Find all `1.3.6` references and replace them with the new version `1.3.7`.
- Find all `1-3-6` references. These will be in links to the release article for the previous version. The link should be replaced with the link to the new article.

Writing release notes

There are two places that changes are documented in the source code. The first is a HTML-based summary that is accessible from the splash screen that ElectrumSV shows when it starts up. The second is the text-based formal `RELEASE-NOTES` file in the top level of the source code.

The HTML-based summary is intended to be a list of user focused descriptions of the main changes in the release. It lists the same changes as those chosen for the release article.

The `RELEASE-NOTES` file is intended to be developer oriented, and should attempt to list all the changes made and included in the release.

4.3.2 Pre-build testing

There are two different kinds of pre-build testing, both manual and automatic. The manual tests are primarily those which involve a user checking the user interface works as it should. The automatic tests ensure the code is correct as it is possible for such a tool to detect, and that when asked to perform processes the outcomes of those processes are as they should be.

User interface testing

There is a checklist of common use cases for ElectrumSV that the user interface is manually stepped through. New accounts are created, keys and seeds are imported, invoices are paid, hardware wallets are plugged in and out and most if not all of the menu options are used in order to ensure they still work.

Note: TODO: Reference manual user interface testing documents.

As bugs, problems or small aspects that can be improved are identified, they are fixed and the relevant user interfaces are retested. Along these lines, if intuitively something does not quite seem like it is working, time is spent to work out why.

Code analysis

As a part of normal development, before code changes are committed to the Github source code repository, developers are expected to run code quality tools. If they push the changes to Github and they have made changes that do not meet code quality standards, then the CI process will do those same checks and error. The changes made to both prepare the release and fix any problems observed in the user interface should be tested by the developer.

mypy

Python is a programming language with optional typing. For users who choose to use typing, this tool can then try and work out if the code that uses those types is buggy or incorrect.

Running mypy on Windows, Linux or MacOS:

```
mypy --config-file mypy.ini --python-version 3.7
```

pylint

This tool checks for general code correctness and common errors, and warns the developer if it finds any.

Running pylint on Windows, Linux or MacOS:

```
pylint --rcfile=.pylintrc electrum-sv electrumsv
```

Unit testing

The existing collection of unit tests ensure that a range of processes work correctly. This includes how the code handles different kinds of accounts, migration of wallets from older versions to newer versions, old Electrum seed words, new Electrum seed words, BIP39 seed words, different key types and so on. Running these against lower level changes can often help detect regressions or oversights made in implementing those changes.

Running the unit tests on Windows:

```
pytest electrumsv\tests
```

Running the unit tests on Linux or MacOS:

```
pytest electrumsv/tests
```

4.3.3 Building the release

The continuous integration (CI) service is hooked up to Github. Every time a set of changes are pushed to Github it automatically triggers the CI to test and build those changes. Every build results in what are called a set of artifacts, which are the executables and archives produced as a result of that build. If the developer adds a Git tag structured in a way to designate a release version to the changes they push, then this modifies the build process and produces an official versioned set of build artifacts.

Tagging the latest code as a potential stable release of a `1.3.7` version:

```
git tag sv-1.3.7
```

The developer than pushes both the latest code and the tag to Github, both separately, and in that order:

```
git push
git push --tags
```

A build is only triggered if unpushed code changes are pushed. And the build only looks for the release tag at the start. So the developer needs to push unpushed code changes, and then the new release tag in quick succession.

Build errors

The build runs all the tests that the developer should run before they push the final changes. If they fail, or their development tools are out of date, this might mean that either the developer did not run the tests correctly or that the developer needs to update their tools.

Recapping the automated tests employed:

- The unit tests.
- The functional tests.
- Pylint for style and correctness checking.
- Mypy for type checking.

If there are build errors or the build needs to be rerun, the developer needs to delete the tag and recreate it, and push a new tag with additional code changes to trigger a new build.

Deleting the local tag for a `1.3.7` release:

```
git tag --delete sv-1.3.7
```

Deleting the remote tag for a `1.3.7` release:

```
git push origin --delete sv-1.3.7
```

Testing the build

Once a successful candidate build has been made, the build artifacts are downloaded. One artifact is deleted, the Windows installer which is named with the `-setup.exe` suffix. At this time we do not support this or test it, and in the longer term we will provide this in the form of a Windows Store application.

The build testing is not extensive. If a build executable runs and the wallet user interface appears, then all testing of both functionality and user interface within the pre-build testing will represent how the build behaves.

Linux

There are no Linux builds at this time, so there is no need for testing at this stage.

Note: If a member of the community creates an AppImage build process that is of sufficient quality, we would be willing to help them maintain it and use it in producing official Linux builds.

MacOS

The build is downloaded to a MacOS device, and run.

The following trivial steps are tested:

1. Funds are sent to the wallet on the MacOS device.
2. The funds are then sent back out to an external wallet.

Windows

There are two builds on Windows, a portable build and a non-portable build. A quick recap on the difference is that the portable build stores it's data in a directory local to the portable build executable. The non-portable build stores it's data in the user's application data directory.

The following trivial steps are tested for the non-portable build:

1. Funds are sent to the wallet on the MacOS device.
2. The funds are then sent back out to an external wallet.

The non-portable build is merely started, and if the user interface appears and the wallet selection screen can be reached, it is deemed sufficient.

4.3.4 Deployment

There are a range of steps to doing the deployment.

Build files

The build files are currently hosted for download on Amazon S3 storage rather than on the web site. This was initially done in order to try and reduce the false positive flagging for Malware that ElectrumSV gets on Windows, because of it's use of Pyinstaller. The process of uploading these is intended to be paranoid to ensure that the files uploaded are the actually the ones the CI process produced.

After the build artifacts are uploaded to Amazon S3 storage, they are re-downloaded and the SHA256 hash of each is compared to those that CI produced by redownloading the build hashes from CI.

Web site

Besides reflecting the latest release, another function of the web site is that it hosts a JSON file with signatures from at least one developer for the given release version and date. This is used by the update checker to alert users that there is a new release. The web site also hosts the GPG signatures from at least one developer, which need to be added before it is generated.

Update signatures

The keys used to verify that a release has been signed by a known developer are hard-coded into each build. This makes it difficult to add new signing developers, as users with older builds will lack the keys for those new developers, those builds will appear illegitimate. It is probably a good idea for the process to change sooner rather than later to prepare for working around this.

One or more of the developers can sign to announce the release of the build, and each should do the following:

1. Take the release version which might be `1.3.7`.
2. Take the release date which might be `2020-10-08T20:00:00.000000+13:00`.
3. Combine them which in this case will result in `1.3.72020-10-08T20:00:00.000000+13:00`.
4. Go into the signing wallet and select the signing key.
5. Select the *Sign/verify message* menu.
6. Enter the combined text.
7. Click the *Sign* button and enter the wallet password.
8. Copy the signature and place in the *release.json* file.

The existing *release.json* file is included in the web site generation content, and should be updated and it will automatically be included in the generated web site.

GPG signatures

In addition to hashes proving the integrity of downloaded build files, there are also GPG signatures that indicate who they came from. The public keys of the developers who might sign the build files are in Github much like the SHA256 hashes for each build file.

A sub-directory should be made within the *download* web site content directory for the release version, and the GPG signatures for each new build file placed in there.

Generation

With GPG signatures and release version signatures in place, and also updated for the new version and build files, the final web site can be generated and put in place on the ElectrumSV web host. The generation instructions documented in the web site directory. Assuming that the developer has already been generating the web site in the past, the following commands are all they need to do one final generation.

```
cd docs
cd website
pelican -s pelicanconf.py
```

Standard deployment steps need to be followed and the new uploaded *html* directory needs to match the existing one in the following ways:

1. The same owner using `chown -R`.
2. The same permissions using `chmod -R`.

Documentation

The documentation is hosted on the Read the Docs service. As changes are pushed to the Github repository, Read the Docs is notified and they fetch the changes and trigger an update of the documentation. This mostly benefits users being able to view development documentation. The deployed documentation for a given release cannot change any time post-release development changes are made.

After the tag for the release changes is pushed to Github, a developer needs to add it to the list of tags that Read the Docs is hosting documentation for. And then they need to make it the default tag so that the documentation URL `electrumsv.readthedocs.io` goes there by default.

Github

At this point the documentation, the web site, and almost all other changes should be present in Github. The one thing that may be missing is the SHA256 hashes for the build files, which need to be added to the file `build-hashes.txt` in the source code, and pushed as well. Beyond that they need to be merged into the master branch, which is the place we recommend users go to find them.

Github releases

Github has it's own system for projects to make releases, and we do use that, but we do not use it to release build files. It's primary used to formally designate the release tag as a new release, and associate it with a list of the changes in the release. The changes listed there are taken directly from the `build-hashes.txt` file.

Release article publication

This should just be a matter of applying any final polish to the already prepared release article and pressing whatever resembles the *Publish* button.

Announcements

The link to the release article should be posted to the following places with some additional decorative text.

- Twitter.
- The Metanet.ICU slack.
- The Atlantistic Unwriter slack.
- Anywhere else.

Note: TODO: Guidelines to how we write the standard decorative text should be added here.

4.3.5 The release checklist

It is not realistic for developers to read this document when they want to make a release and step through the description of the process. Instead, they should refer to the following checklist and where necessary refer to the description of the process for context and further details.

Note: TODO: Formalise the above as a list of concrete steps.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex