Ethereum SLIP-39 Account Generation

Perry Kundert

2021-12-20 10:55:0

Creating Ethereum accounts is complex and fraught with potential for loss of funds.

Creating a BIP-39 seed recover phrase helps, but a **single** lapse in security dooms the account. If someone finds your recover phrase, the account is gone.

The SLIP-39 standard allows you to split the seed between 1 or more groups of multiple recovery phrases. This is better, but creating such accounts is difficult; presently, only the Trezor supports these, and they can only be created "manually". Writing down 5 or more sets of 20 words is difficult and time consuming.

The python-slip39 project exists to assist in the safe creation and documentation of Ethereum HD Wallet accounts, with various SLIP-39 sharing parameters. It generates the new wallet seed, generates standard Ethereum account(s) (at derivation path `m/66'/40'/0'/0/0` by default) with Ethereum wallet address and QR code, produces the required SLIP-39 phrases, and outputs a single PDF containing all the required printable cards to document the account.

On an secure (ideally air-gapped) computer, new accounts can safely be generated and the PDF saved to a USB drive for printing (or directly printed without the file being saved to disk.)

Contents

1 Security with Availability

A 128-bit random "seed" is the source of an unlimited sequence of Ethereum HD Wallet accounts. Anyone who can obtain this seed gains control of all Ethereum accounts derived from it, so it must be securely stored.

Losing this seed means that all of the HD Wallet accounts are permanently lost. Therefore, it must be backed up reliably, and be readily accessible.

Therefore, we must:

- Ensure that nobody untrustworthy can recover the seed, but
- Store the seed in many places with several (some perhaps untrustworthy) people.

How can we address these conflicting requirements?

1.1 Shamir's Secret Sharing System (SSSS)

Satoshi Lab's (Trezor) SLIP-39 uses SSSS to distribute the ability to recover the key to 1 or more "groups".

First, the key material is split between 1 or more groups, and a "group_threshold" of how many groups must bet successfully collected to recover the key.

For example, you might have First, Second, Fam and Frens groups, and decide that any 2 groups can be combined to recover the key. Each group has members with varying levels of trust, so have different number of Members, and differing numbers Required to recover that group's data:

Group	Required	Members	Description
First	1	1	Stored at home
Second	1	1	Stored in office safe
Fam	2	4	Distributed to family members
Frens	3	6	Distributed to close friends

The account owner might store their First and Second group data in their home and office safes. These are 1/1 groups (1 required, and only 1 member, so each of these are3 1-card groups.)

If the account needs to be recovered, collecting the First and Second cards from the home and office safe is sufficient to recover the seed, and re-generate the HD Wallet accounts.

Only 2 Fam member's cards must be collected to recover the Fam group's data. So, if the HD Wallet owner loses their home and First group card in a fire, they could get the Second group card from the office safe, and 2 cards from Fam group members, and recover the wallet.

If catastrophe strikes and the owner dies, and the heirs don't have access to either the First (at home) or Second (at the office), they can collect 2 Fam cards and 3 Frens cards (eg. at the funeral?), completing the Fam and Frens groups' data, and recover the HD Wallet account.

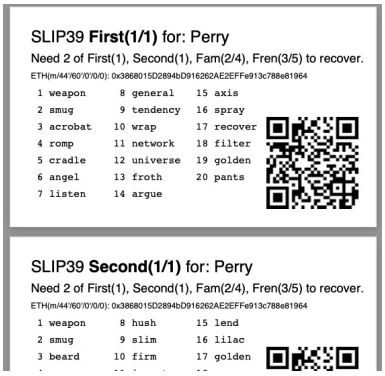

Figure 1: SLIP-39 PDF

2 SLIP-39 Account Generation

Generating a new SLIP-39 encoded Ethereum wallet is easy, with results available as PDF or text. The default groups are as described above. Run the following to obtain a PDF file similar to this example, insert a USB drive to collect the output, and run:

```
$ cd /Volumes/USBDRIVE/
$ python3 -m pip install slip39
$ python3 -m slip39 Perry # or just "slip39 Perry"
...Output SLIP39-encoded wallet for 'Perry' to:\
  Perry-2021-12-22+15.45.36-0x3868015D2894bD916262AE2EFFe913c788e81964.pdf
```

The resultant PDF will be output into the designated file.

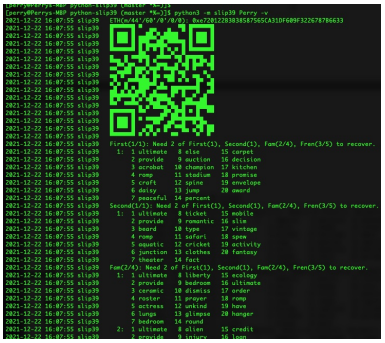

Figure 2: SLIP-39 text

This PDF file can be printed on 3x5 cards, or on regular paper or card stock and the cards can be cut out.

To get the data printed on the terminal as in this example (so you could write it down on cards instead), add a `-v`.

3 Dependencies

Internally, python-slip39 project uses Trezor's python-shamir-mnemonic to encode the seed data, and the Ethereum project's eth-account to convert seeds to Ethereum accounts.

3.1 The `python-shamir-mnemonic` API

To use it directly, obtain , and install it, or run `python3 -m pip install shamir-mnemonic`.

```
$ shamir create custom --group-threshold 2 --group 1 1 --group 1 1 --group 2 5 --group 3 6
Using master secret: 87e39270d1d1976e9ade9cc15a084c62
Group 1 of 4 - 1 of 1 shares required:
merit aluminum acrobat romp capacity leader gray dining thank rhyme escape genre havoc furl breathe class pitch location render
Group 2 of 4 - 1 of 1 shares required:
merit aluminum beard romp briefing email member flavor disaster exercise cinema subject perfect facility genius bike include say
Group 3 of 4 - 2 of 5 shares required:
merit aluminum ceramic roster already cinema knit cultural agency intimate result ivory makeup lobe jerky theory garlic ending s
merit aluminum ceramic scared beam findings expand broken smear cleanup enlarge coding says destroy agency emperor hairy device
merit aluminum ceramic shadow cover smith idle vintage mixture source dish squeeze stay wireless likely privacy impulse toxic mo
merit aluminum ceramic sister duke relate elite ruler focus leader skin machine mild envelope wrote amazing justice morning voca
merit aluminum ceramic smug buyer taxi amazing marathon treat clinic rainbow destroy unusual keyboard thumb story literary weapo
Group 4 of 4 - 3 of 6 shares required:
merit aluminum decision round bishop wrote belong anatomy spew hour index fishing lecture disease cage thank fantasy extra ofter
merit aluminum decision scatter carpet spine ruin location forward priest cage security careful emerald screw adult jerky flame
merit aluminum decision shaft arcade infant argue elevator imply obesity oral venture afraid slice raisin born nervous universe
merit aluminum decision skin already fused tactics skunk work floral very gesture organize puny hunting voice python trial lawsu
merit aluminum decision snake cage premium aide wealthy viral chemical pharmacy smoking inform work cubic ancestor clay genius f
merit aluminum decision spider boundary lunar staff inside junior tendency sharp editor trouble legal visual tricycle auction gr
```

3.2 The `eth-account` API

To creete Ethereum accounts from seed data, two steps are required.

First, derive a Private Key from the seed data plus a derivation path:

```
>>> seed=codecs.decode("dd0e2f02b1f6c92a1a265561bc164135", 'hex_codec')
>>> key=eth_account.hdaccount.key_from_seed(seed, "m/44'/60'/0'/0/0")
>>> keyhex=codecs.encode(key, 'hex_codec')
>>> keyhex
b'178870009416174c9697777b1d94229504e83f25b1605e7bb132aa5b88da64b6'
```

Then, use the private key to obtain the Ethereum account data:

```
>>> keyhex.decode('ascii')
'178870009416174c9697777b1d94229504e83f25b1605e7bb132aa5b88da64b6'
>>> keyhex = '0x'+keyhex.decode('ascii')
>>> keyhex
'0x178870009416174c9697777b1d94229504e83f25b1605e7bb132aa5b88da64b6'
>>> account = eth_account.Account.from_key(keyhex)
>>> account
<eth_account.signers.local.LocalAccount object at 0x7fba368ae670>
>>> account.address
'0x336cBeAB83aCCdb2541e43D514B62DC6C53675f4'
```