Non-EVM networks

You can manage users' non-EVM accounts and assets with their approval, usingSnaps API methods.

important Managing users' keys comes with great responsibility: Misplaced or stolen private keys can lead to a complete loss of funds for users of your Snap.

Responsible key management

It's critical to practice responsible key management. The general rule is:Don't create a situation where your users can lose assets.

examples of responsible key management: * Deriving private keys and/or storing them in Snaps persistent storage, without ever moving them * out of theSnaps execution environment * . * Ensuring arbitrary code execution can't access irreversible operations or private keys. * Asking the user for consent and informing them of what's going to happen before performing * irreversible operations. examples of irresponsible key management: * Allowing extraction of private keys outside the Snap in any way, especially through RPC or * network connections. * Executing arbitrary or untrusted code with access to private keys. * Not getting properly informed consent before performing irreversible operations. * Asking for consent but ignoring the decision. * Exposing key material in clear-text. * Producing a bug that leads to any of the above.

Derive keys

To derive a user's private keys:

1. Choose between the BIP-32 or BIP-44 specifications to derive the user's private keys.
2. If the keys you want to derive conform to theBIP-44
3. structure, usesnap_getBip44Entropy
4. to derive them.
5. Otherwise, usesnap_getBip32Entropy
6. .
7. Add the required permission to your manifest file.
8. Find out the derivation path to use.
9. This is dependent on the application you're building.
10. Use the@metamask/key-tree
11. module to derive the keys.
12. Any additional code, for example, to derive addresses from keys, is application-specific.

Dogecoin example

For example, to derive Dogecoin keys:

1. Dogecoin uses the BIP-44 scheme, so you'll usesnap_getBip44Entropy
2. .
3. Dogecoin has coin type3
4. , so add the following to the manifest file:
5. snap.manifest.json
6. {
7. "initialPermissions"
8. :
9. {
10. "snap_getBip44Entropy"
11. :
12. [
13. {
14. "coinType"
15. :
16. 3
17. }
18.]
19. }
20. }
21. Dogecoin uses the following derivation path:
22. m/44'/3'/0'/0/{address_index}
23. To get the second Dogecoin account, add the following code to your Snap:
24. index.js
25. import
26. {

```
getBIP44AddressKeyDeriver
}
from
"@metamask/key-tree"
;
// Get the Dogecoin node, corresponding to the path m/44'/3'.
const
dogecoinNode
=
await
snap
.
request
(
{
method
:
"snap_getBip44Entropy"
,
params
:
{
coinType
:
3
,
}
,
}
)
;
/**
```

- Create a function that takes an index and returns an extended private key for m/44'/3'/0'/0/address_index.

```
```

- The second parameter to getBIP44AddressKeyDeriver isn't passed. This sets account and changes to 0.

```
*/
const
deriveDogecoinAddress
=
await
getBIP44AddressKeyDeriver
(
dogecoinNode
)
;
// Derive the second Dogecoin address, which has index 1.
const
secondAccount
=
deriveDogecoinAddress
(
1
)
;
```

Examples

The following are examples of existing Snaps that manage accounts and keys:

- Dogecoin Snap tutorial
- Consensys StarkNet Snap
- Account Labs Bitcoin Snap
- @metamask/bip32-example-snap
- @metamask/bip44-example-snap

Edit this page