Kusama

Kusama is a "canary network" for Polkadot, an early unaudited release of the code that is available first and holds real economic value. For developers, Kusama is a proving ground for runtime upgrades, on-chain governance, and parachains.

No Promises.

What can you do on Kusama?

On Kusama you can:

- Explore the politics. Campaign as a councillor or vote for new runtime proposals using Democracy.
- Hone your validator set-up. The minimum requirement for staking as a validator on Kusama is much lower than it is expected for Polkadot. There are also programs in place such as Thousand Validators to help community validators rise the ranks.
- Deploy a parachain. Kusama will get the functionality required for parachains before Polkadot. This includes participating in a parachain slot auction and composable applications.
- Join a cyber secret society. The Kappa Sigma Mu fraternity asks you to get a Kusama tattoo to join.
- Expect Chaos. Kusama is getting more battle tested day-by-day but it's built on the foundations of moving fast and trying new things.

Kusama is owned by those who hold the Kusama tokens (KSM). There's no central kill switch and all changes are made through the protocol's on-chain governance. Kusama is experimental. There are no guarantees.

Who can participate on Kusama?

The short answer is, everyone!

Kusama is a permissionless network and anyone can come along and start using it.

Those who participated in the Polkadot sale can claim a proportional amount of KSM through the Kusama Claims process.

KSM used to be publicly available through a highly-frictional faucet, but this has been decommissioned. Web3 Foundation is considering new ways to distribute KSM for people who need KSM to build.

As a KSM holder, you are able to interact with all the features of the Kusama network such as staking (i.e. validating or nominating), governance, parachain auctions, basic transfers and everything else.

Contact

General

- Kusama forum
- Kusama chat
- support@kusama.network *Email support*

Get help claiming KSM

- KSM Claims Support chat

Validator chat

- Validator chat (English)
- To join the Chinese Validator chat, message Anson

Report an issue or submit bugs

- sos@kusama.network

Get updates

- @kusamanetwork
- Kusama newsletter
- r/kusama

Getting Started

It's time to get started on the Kusama network! Please check out our recommended list for getting started.

For brand-new learners of Blockchain technology:

- The Blockchain Fundamentals MOOC course is a great introduction to start familiarizing yourself with blockchain concepts such as cryptography and networks, and how these play into things like decentralization and cryptocurrency.

  This is recommended for users with backgrounds of all levels, and the course is free!

For brand-new learners of Kusama, check out:

- Reading the Kusama Overview page
- Reading the "What is Kusama" article on Medium.
- Creating a Kusama Account
- Using Kusama endpoints
- Claiming Kusama KSM tokens
- Running a Validator on Kusama
- Staking as a Nominator on Kusama

For brand-new learners of Kusama's cousin network, Polkadot, please head over to our Polkadot Wiki.

While Kusama does not support smart contracts natively, building apps on it is still possible (example: RMRK.app). If you're interested in diving deeper into *proper* development, however, the builders guide from Polkadot's wiki applies to Kusama as well.

Kusama Faucet

Due to high demand and spamming issues, the Kusama faucet is no longer in operation. The Web3 Foundation has considered new ways to distribute KSM for people who need KSM to build.

If you are interested in obtaining KSM for building or research, you can apply through the Treasury or receive a tip for doing something cool in the community.

Kusama Claims

The Kusama network is Polkadot's experimental, community-focused R&D network. If you held the DOT indicator token at the time of the genesis block of Kusama, you are entitled to claim an equivalent amount of KSM on the Kusama network.

You can claim KSM by signing a message with the Ethereum account that holds your DOT indicator tokens. There is no deadline for claiming KSM.

Step 1. Create a Kusama account

You will need to generate a Kusama account to claim KSM. There are a few ways you can create one.

For most users, we recommend using the Polkadot{.js} browser extension since it will allow you to store your encrypted keystore locally.

In terms of hardware wallet support, you can use the Kusama Ledger application, or Parity Signer.

Another option you may consider using is the Subkey command line utility, which will allow you to take extra steps to protect the security of your key.

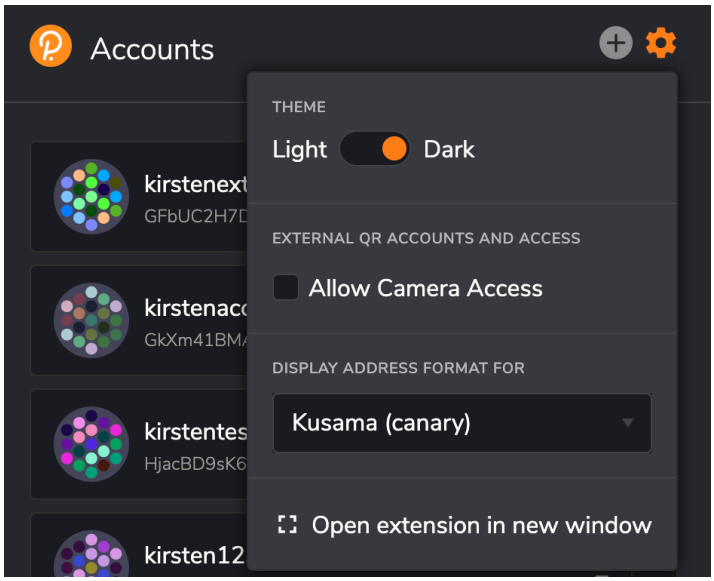
There are a variety of other wallets that you can use; see the Polkadot Wiki Wallets page. Many of these wallets support generating Kusama accounts as well.

Using Kusama Ledger application

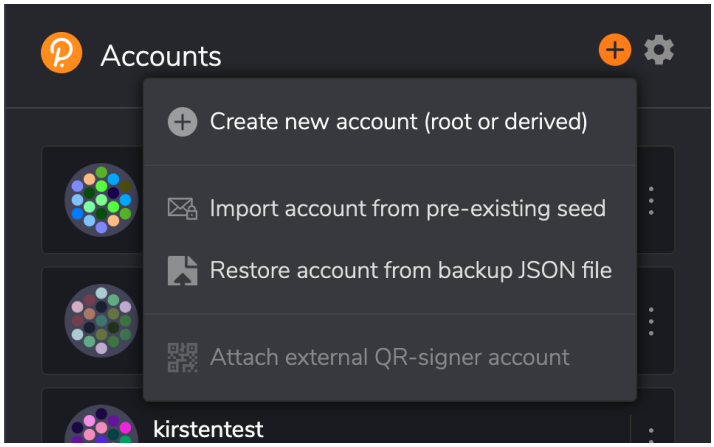
Please follow the instructions for installing the Kusama Ledger application and then proceed to step 2 below.

Using Polkadot{.js} extension (Chrome/Brave or Firefox)

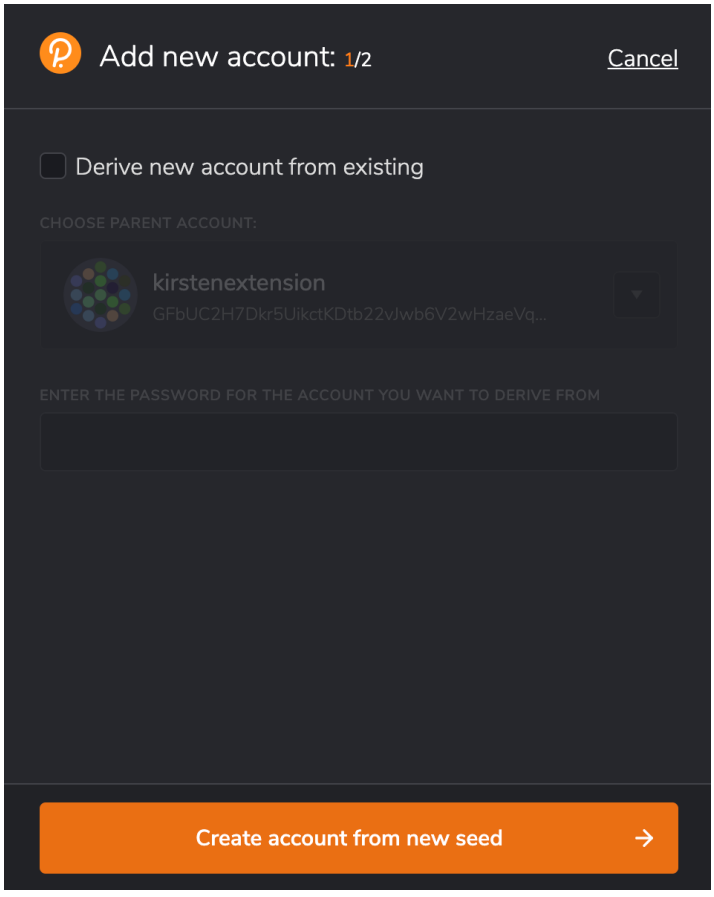
1. Install the Polkadot{.js} extension from the Chrome store or Firefox store.
2. Click on the settings button to change the network to "Kusama (canary)".



3. Create a new account by clicking on "Create new account (root or derived)".

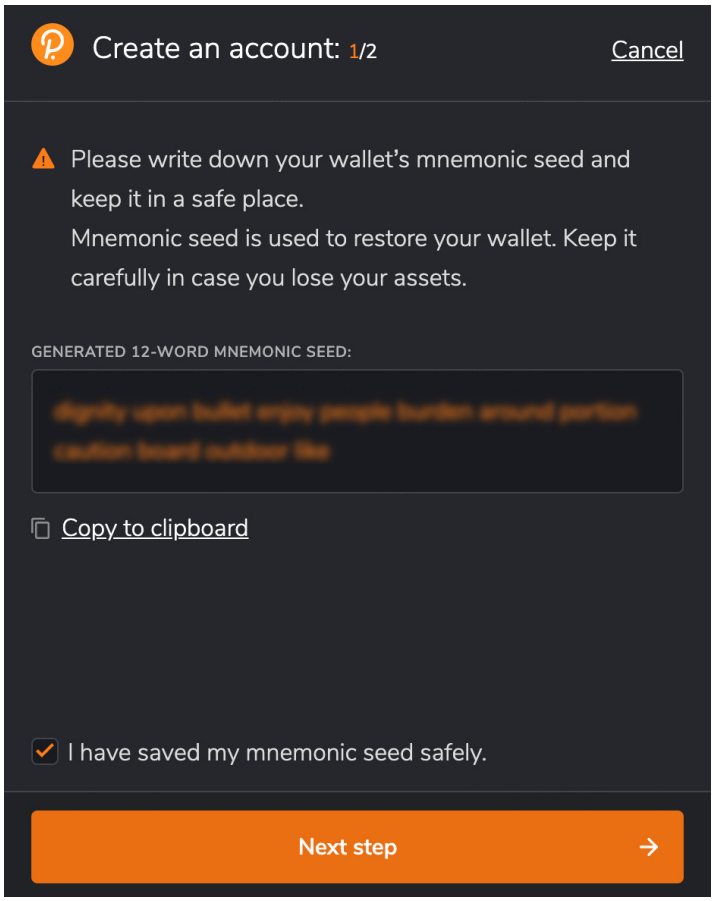


4. Uncheck the option to create an account dervied from another seed.

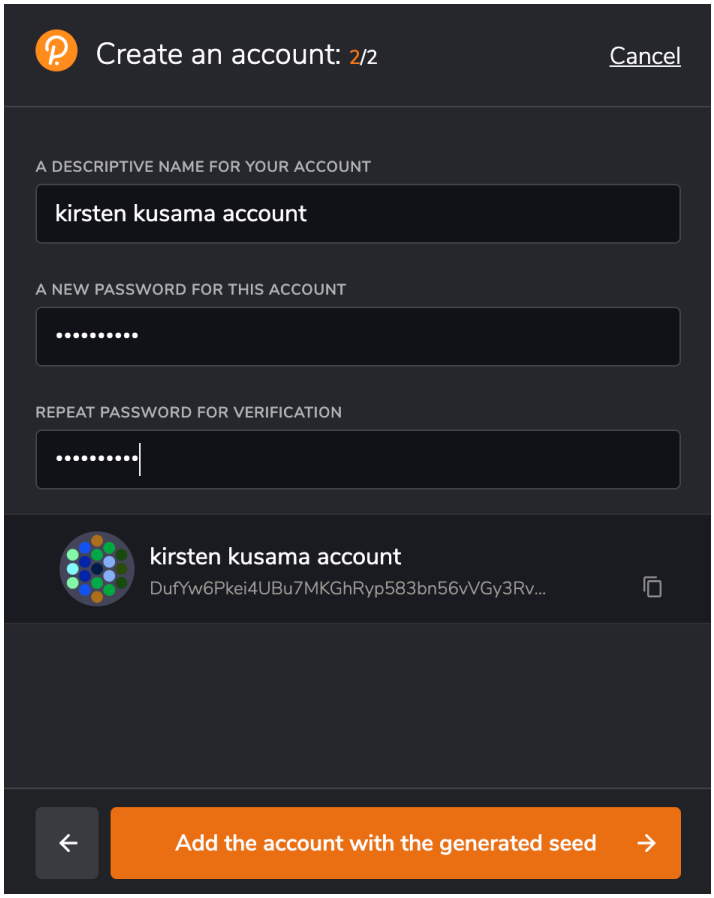


5. Copy the seed phrase and store it somewhere safe. Don't share the seed phrase with anyone, you can use it to access your account if you forget your password or want to import your

account again.



6. Enter a name for the account and type a strong password (at least 6 characters).



7. Click on "Add the account with the generated seed".
8. You can copy the account's address to the clipboard by clicking on its identicon.

Using Subkey

Installation

You can install Subkey with this one-line command:

```
cargo install --force --git https://github.com/paritytech/substrate subkey
```

Note that you will already have had to install the proper Rust version and dependencies. If you have not done so, or experience problems installing using that command, run the following commands first, and then re-try the previous command:

```
curl https://sh.rustup.rs -sSf | sh

rustup update nightly
rustup target add wasm32-unknown-unknown --toolchain nightly
rustup update stable
cargo install --git https://github.com/alexcrichton/wasm-gc
```

Alternatively, you can build Subkey from the source code.

1. Follow the build instructions for Substrate.
2. When building, only build Subkey by typing `cargo build -p subkey`.
3. The executable is `./target/debug/subkey`.

Usage

You can use Subkey on a computer that is not connected to the internet for added security.

The command `subkey --network kusama generate` will generate a new key-pair. If you want to be more secure, use 24 words, `subkey --network kusama generate --words 24`.

```
$ subkey --network kusama generate
Secret phrase `lobster flock few equip connect boost excuse glass machine find
wonder tattoo` is account:
  Secret seed: 0x95b90eb1344e3aea40f4a6dc81622901a2ac39efb331c41db10c311bb9b46927
  Public key (hex):
0xfe7fce341ff73e1db537daa4cc8c539997a8b0654b06cb81c47e4f067f55a65a
  Address (SS58): JL1eTcbzuZP99FjeySkDrMygNREPdbhRyV7iD5AsV4fDRcg
```

The `Address (SS58)` field is what you should use to claim your KSM tokens. Never share your `Secret phrase` or `Secret seed`, as these can both control your funds.

NOTE: Previous versions of Subkey only generated Substrate addresses. If you do not want to generate a new seed, you can convert the Substrate address to a Kusama address by following this

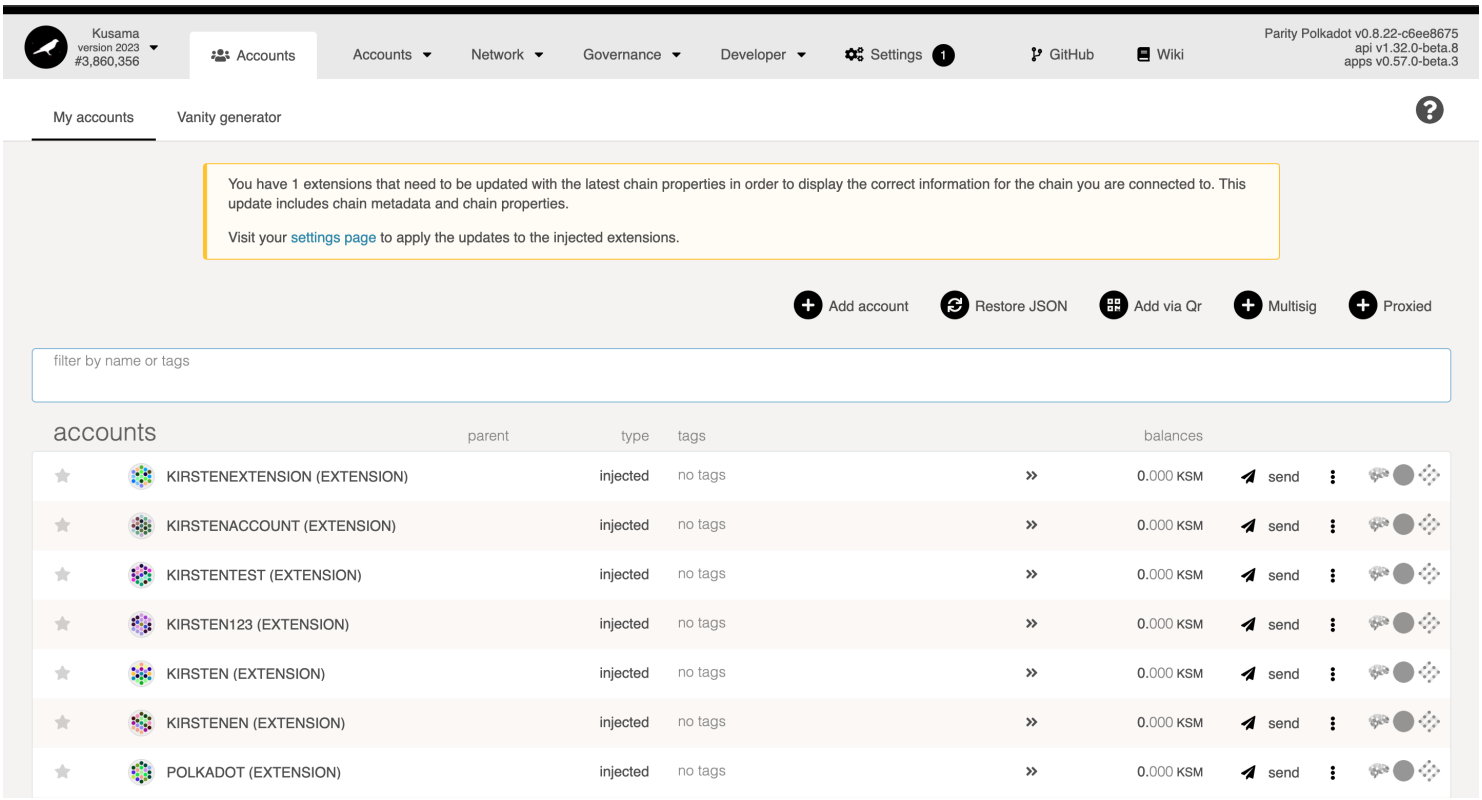
section.

See the Subkey documentation or enter `subkey --help` for more usage examples.

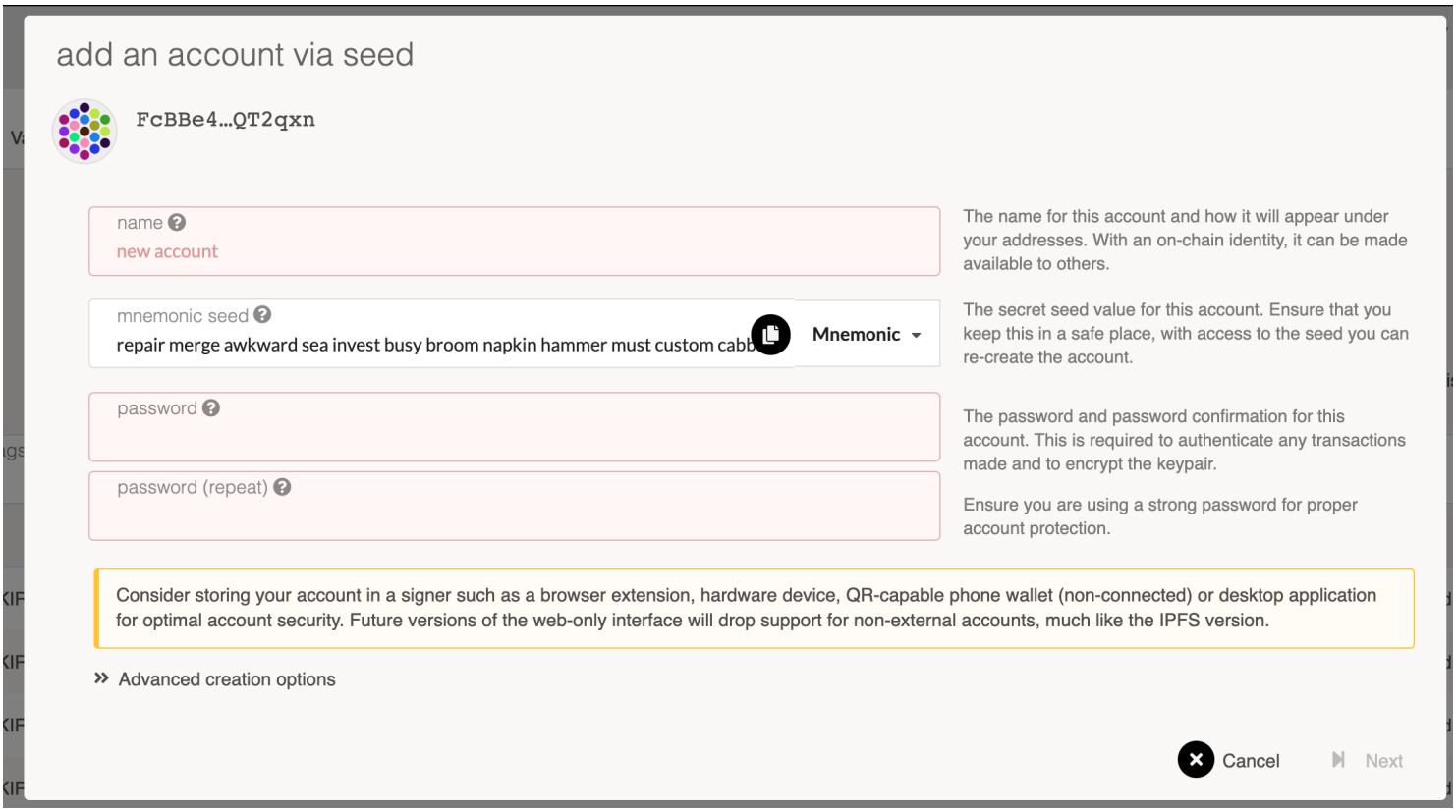
Using Polkadot-JS UI

1. Open up the Polkadot-JS UI and navigate to the top left corner of the navigation. This will open up a panel of network options to select from. Select on "Kusama", either from Parity or Web3 Foundation, then "Switch".

2. Navigate to the Polkadot-JS UI Accounts Tab and click on the "Add account" button.

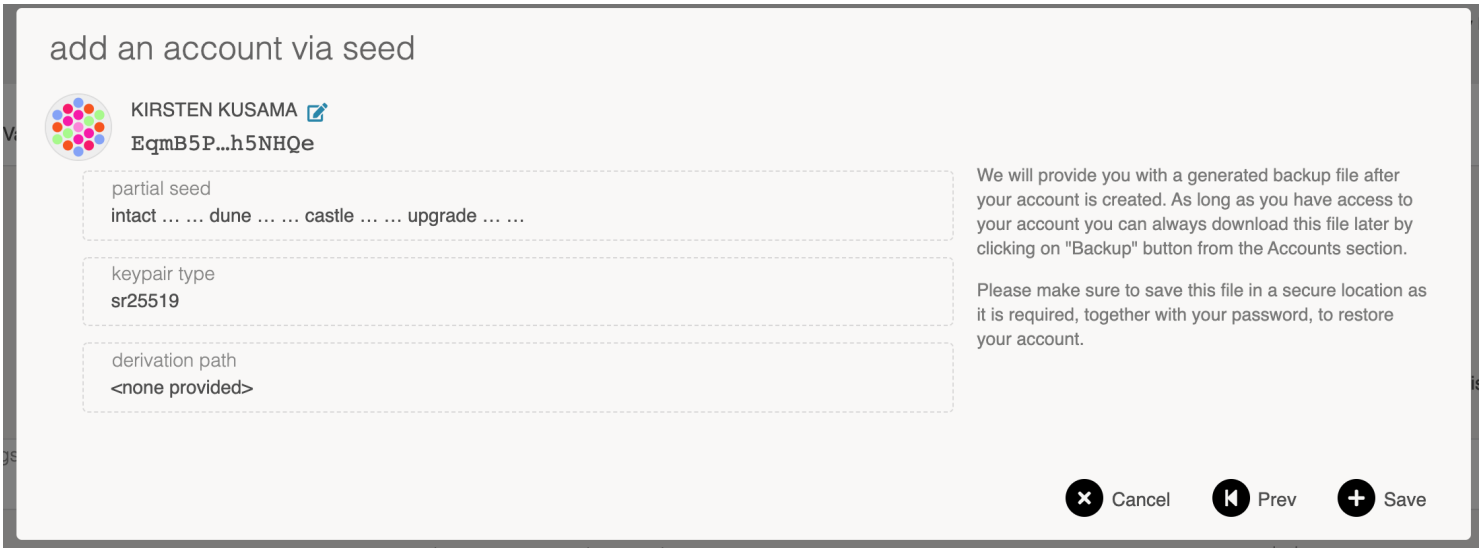


3. Enter a name for your account and create a secure password. This password will be used to decrypt your account. The required text fields to complete are highlighted in pink.



4. Ignore the advanced options unless you want to change the type of cryptography used for your keys (we recommend "Schnorrkel (sr25519)"). You will have to enter an Account Name and a password to protect your account. Be sure to select a secure and hard-to-guess password. Note that anything will be accepted as a password here. Please note: There are no checks to see if it is long enough or secure. You will need this password for any future interaction with or transaction from this account.

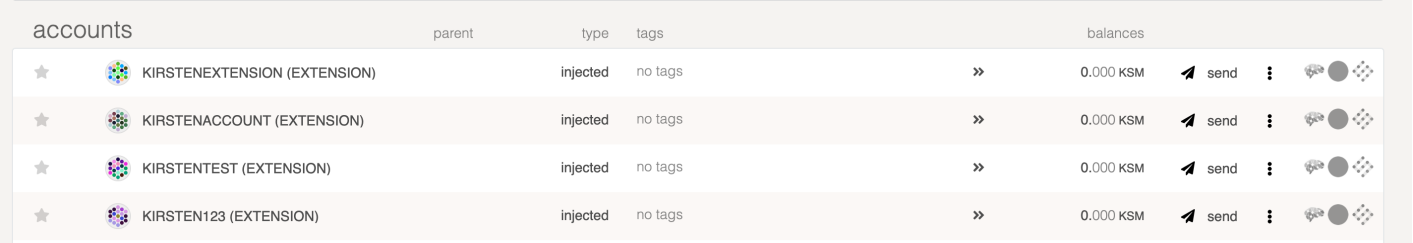
5. Click "Save" and "Create and backup account".



6. Save your encrypted keystore locally. Ideally, you would also save it on an external hard drive or thumb drive, or print it out and be able to re-enter it later. You should not store it in cloud storage, email it to yourself, etc. You can use this backup file to restore your account. The seed in the backup file is not readable unless it is decrypted with the password.

7. The account now appears in your Accounts tab and is backed up to the keystore you just saved.

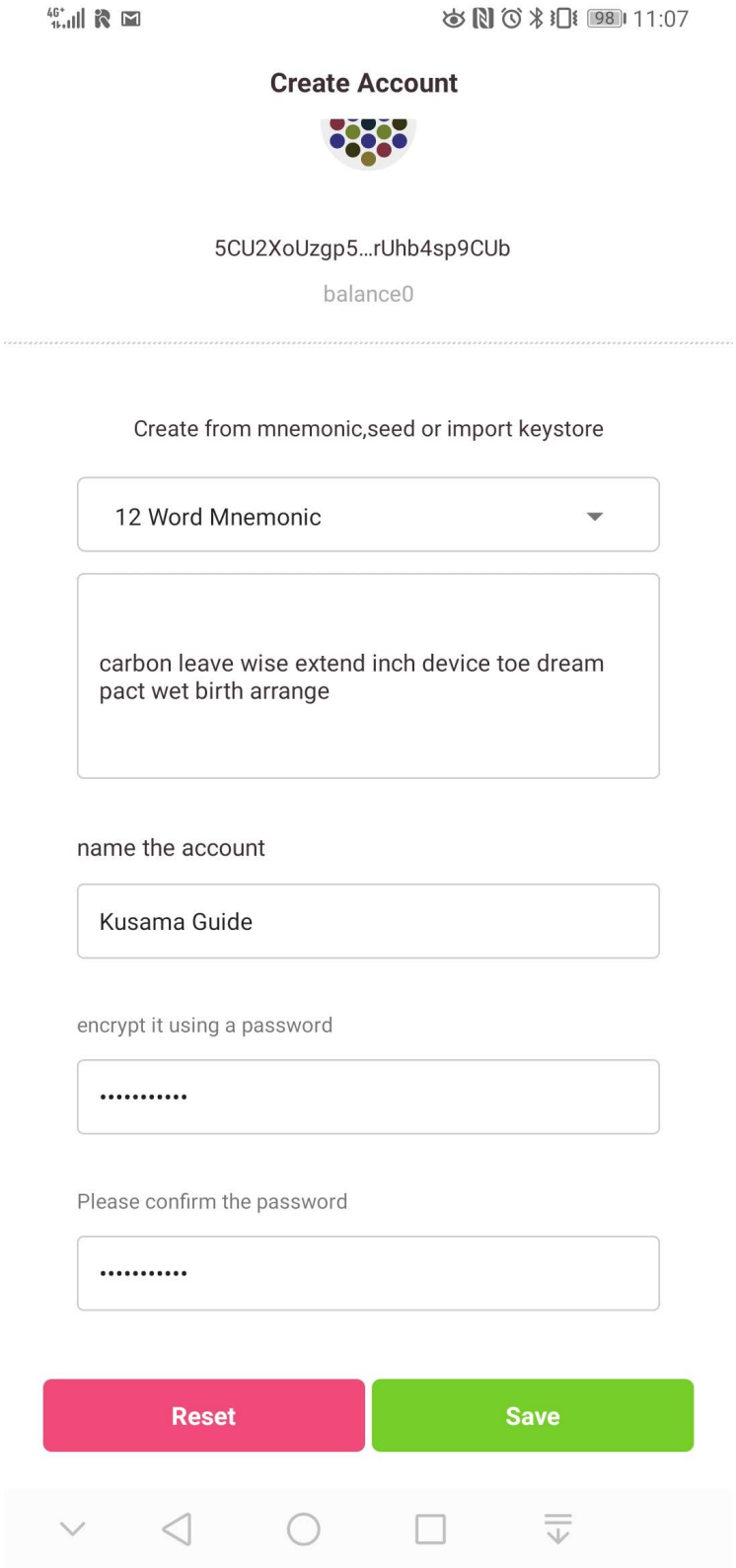
8. Click on the DOT identicon to copy the address to the clipboard.



Using Polkawallet

1. Install Polkawallet. Click "Download" and select the link corresponding to the platform you are using. On Android you may need to allow installing apps from external sources. On iOS, you may need to "trust" Polkawallet in the "General > Profiles & Device Management > Enterprise App" section before running the app.

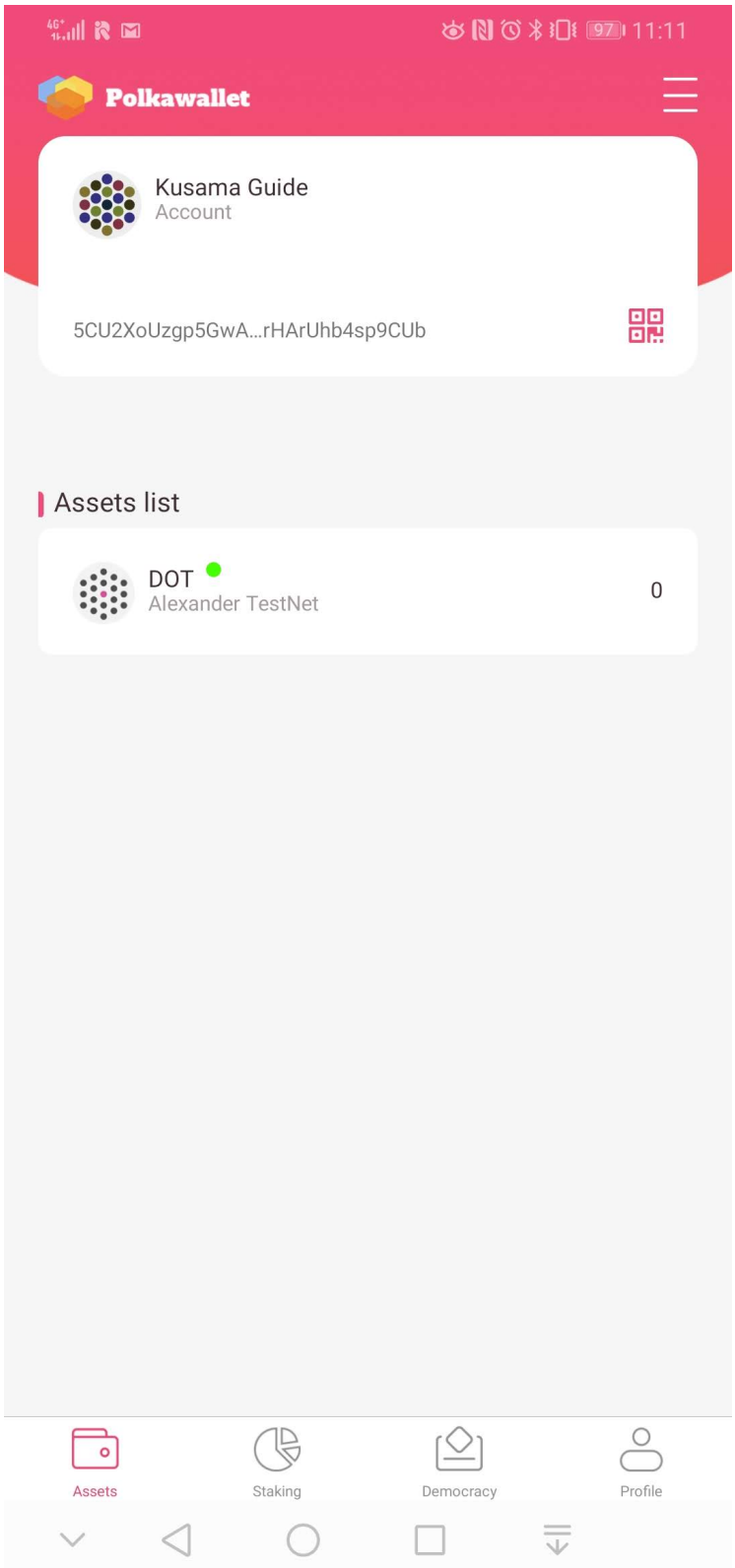
2. Once the app is open, copy the seed phrase and store it in a safe place. Don't share the seed phrase with anyone, you can use it to access your account if you forget your password or otherwise lose your keystore.

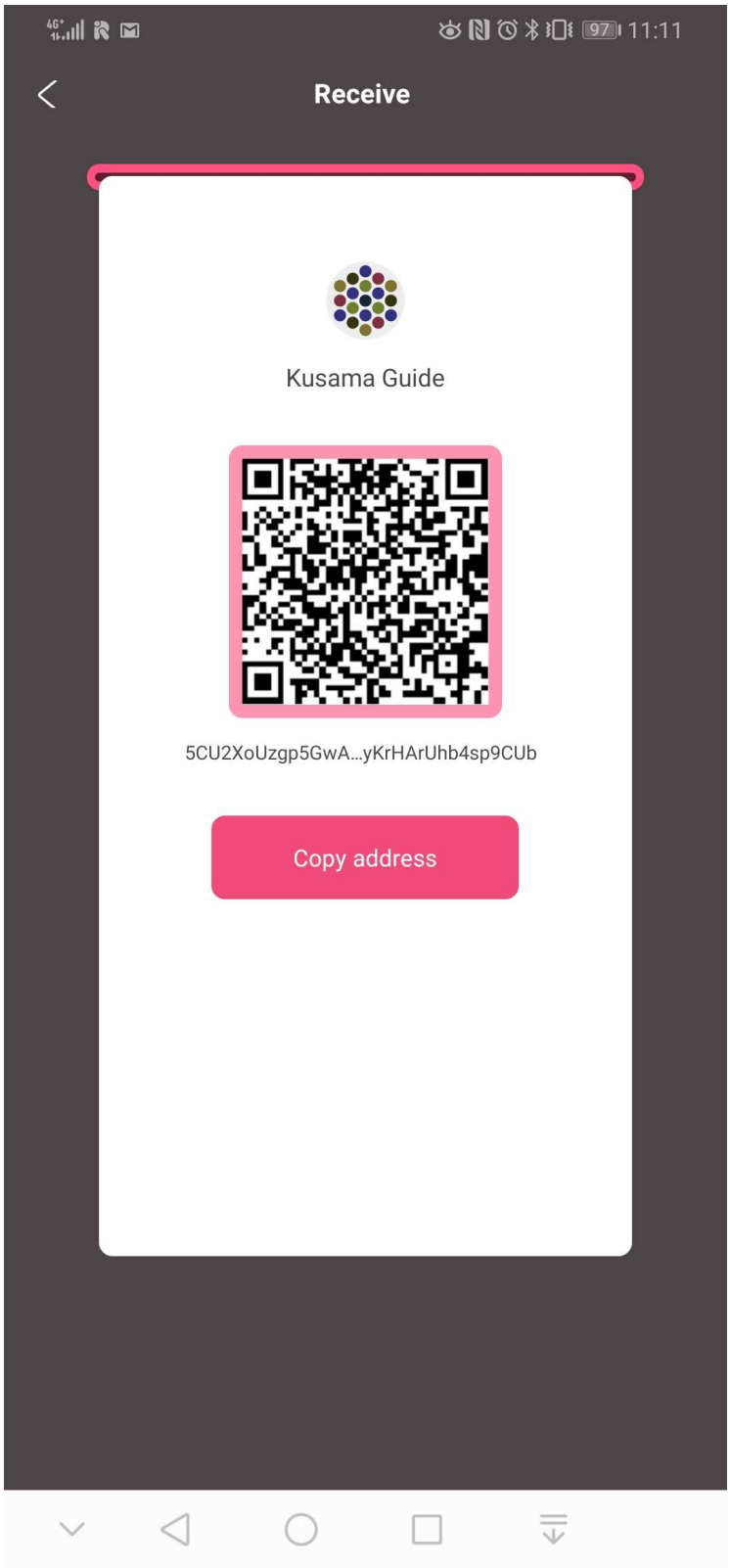


3. Name your account and make a strong password, make sure to write it down in another place, then click "Save".

4. You will be asked to confirm your seed phrase - this is to make sure you have copied it somewhere safe.

5. Click on the pink QR Code symbol and select "Copy address" to copy your address to clipboard.


11:11
Polkawallet
Kusama Guide
Account
5CU2XoUzgp5GwA...rHArUhb4sp9CUb
Assets list
DOT
Alexander TestNet
0
Assets
Staking
Democracy
Profile



6. Get the Kusama address from the Substrate address.

Kusama from Substrate address

If you used one of the generation methods that gave you a generic Substrate address (begins with a `5`), then you will need to take an extra step to turn this into the properly encoded Kusama address.

1. Copy your Substrate generic address to the clipboard.
2. Go to the Polkadot-JS UI.
3. Go to the "Settings" tab and find the configuration for "address network prefix".

4. Select "Substrate (development)" and click "Save and reload".
5. Go to the "Address book" and click the "Add contact" button.
6. Enter your address and give it a name like "My Address".
7. Go back to the "Settings" tab and select the "Kusama (canary)" option in "address network prefix" and click "Save and reload".
8. Go back to the "Address book" and find the account you just added (it will have the same name).
9. The address is now formatted as a Kusama address.

Step 2. Get KSM tokens

There are two methods to claim KSM.

DOT Holders

Those who participated in the Polkadot sales before 2020 and have been allocated DOT indicator tokens on Ethereum can claim a proportional amount of KSM on the Kusama Network.

To do this you must sign a message containing the address of your Kusama account. You can do this by using the Polkadot-JS UI Claims app.

Generate a Kusama address

If you haven't already done so, you will need to generate a Kusama address. See step 1 above for detailed instructions first.

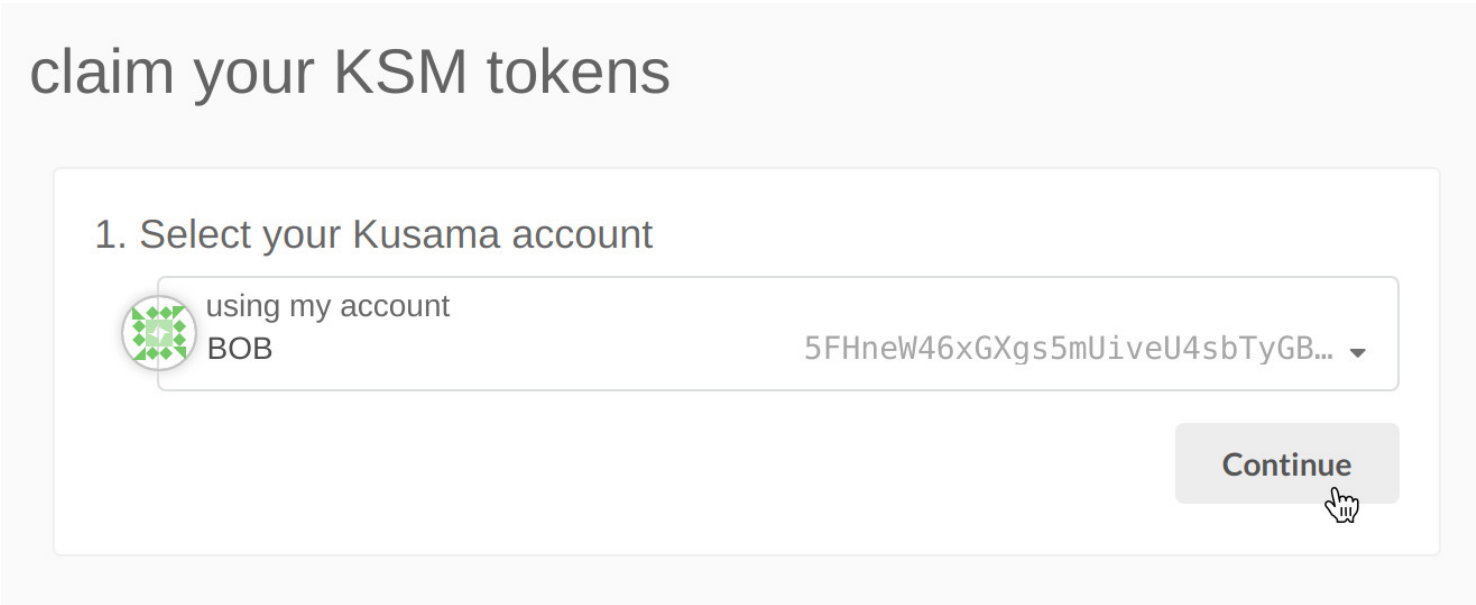
Claiming your KSM with MyCrypto

The Polkadot-JS Claims app helps you sign a message from MyCrypto. MyCrypto is good to use in case you have stored the key to the Ethereum account holding your DOT indicator tokens on a hardware device like a Ledger Nano S or a Trezor. It also supports raw private keys, mnemonics and the Parity signer.

> NOTICE: It is much more secure to download and use the MyCrypto app locally. Please make sure to download the latest version for your operating system. You can always find the most up-to-date releases of the desktop app on their releases page.

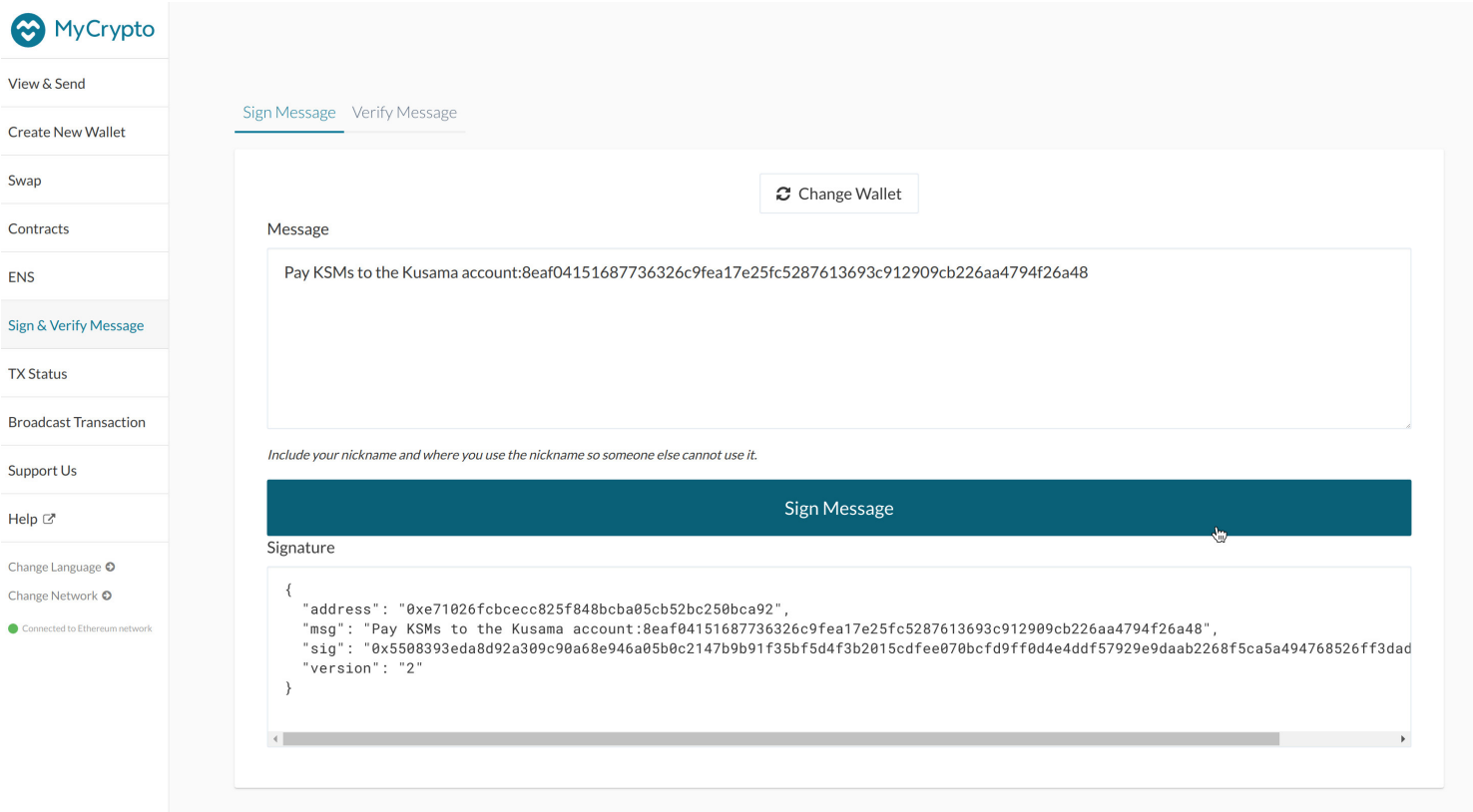
Once you've downloaded MyCrypto and have it running locally (we recommend an air-gapped computer for maximum security), you can start by navigating to the Claims app on the Polkadot-JS

UI. Select the account you would like to claim the KSM into and click the blue "Continue" button to proceed. Your screen should look something like this:



The hex encoded string that follows the sentence: "Pay KSM to the Kusama account:" is the hex-encoded public key of your Kusama account, minus the `0x` prefix. To verify that the public key is correct you can use the Subkey tool to inspect your address.

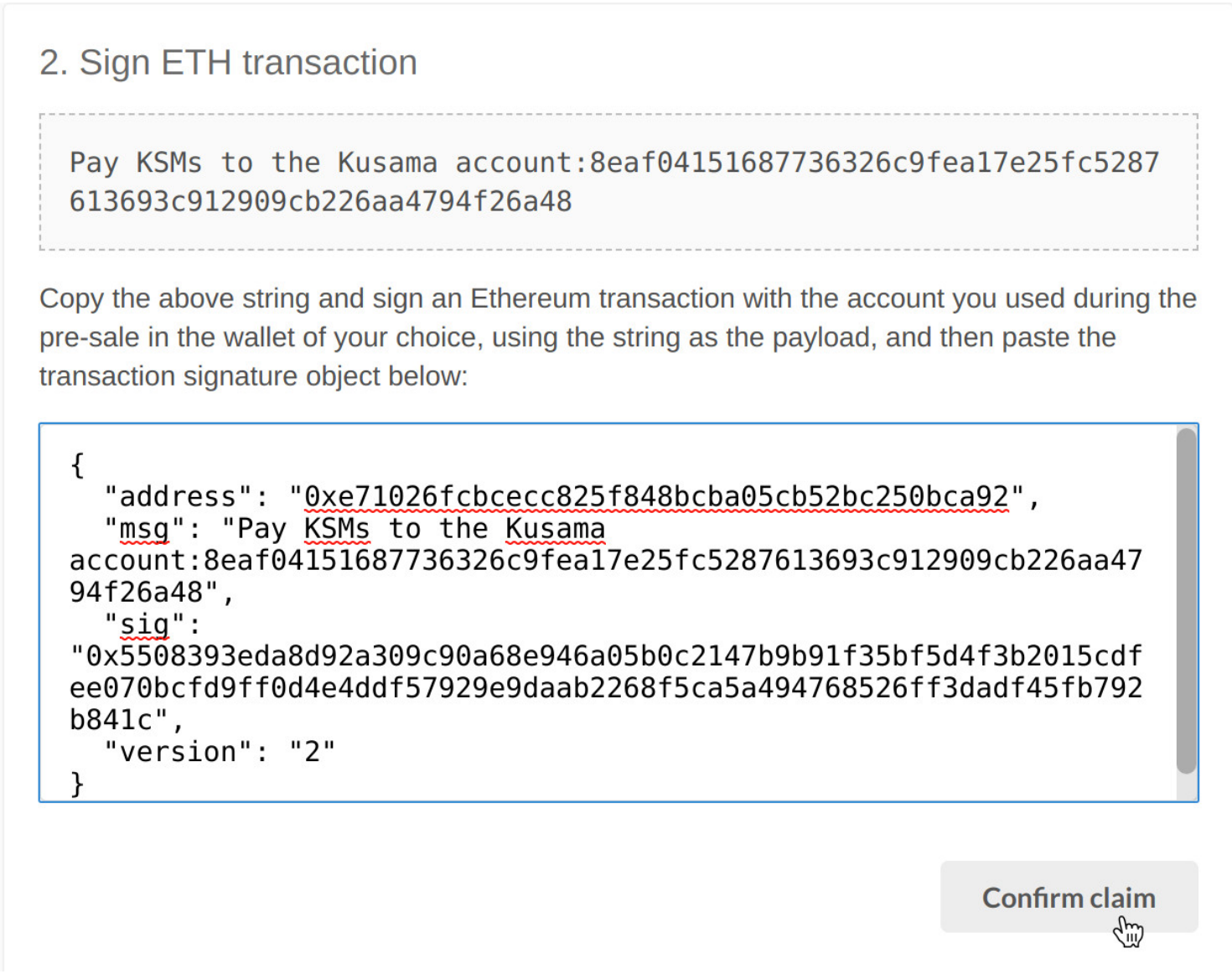
The next step is to go to the MyCrypto application and click on "Sign & Verify Message" tab. This will prompt you to select a method for unlocking your wallet. After unlocking your wallet, you will copy and paste the outputted sentence into the input box.



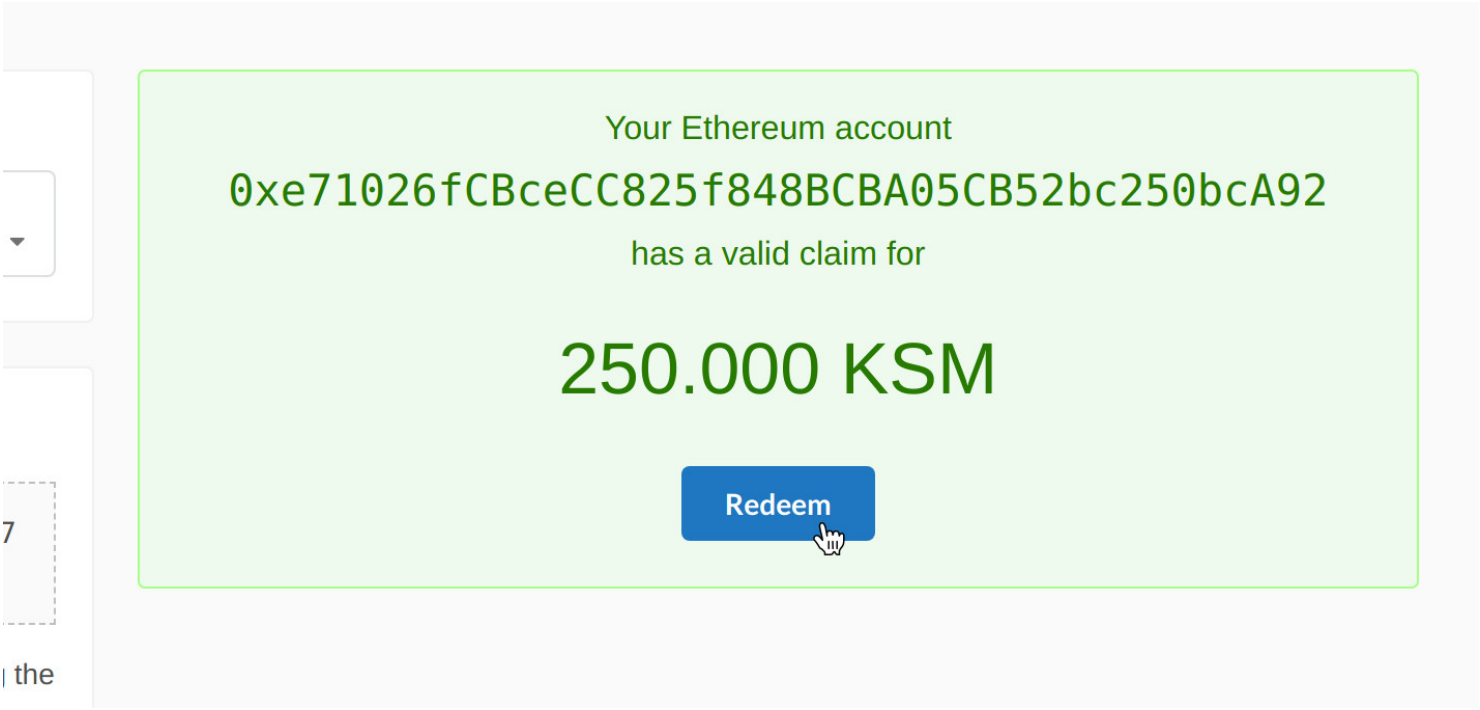
When you click "Sign Message" you will get a JSON output like the below:

```
{
  "address": "0x17aa69b67fba1febbe1254207394ca3875ab7b5f",
  "msg": "Pay KSMs to the Kusama account:28c2abc482a59a82ca6fdb4609bf37f34cb70cf3926da054e25ef3ead02e62
  "sig": "0xfdd522c00791838303d48fe1da9fe8a070d42ce844fa46da7f556b69f477804c6c0fe6a3260ee354009662bc829
  "version": "2"
}
```


Copy and paste the JSON output of the signed message from MyCrypto into the input box on the Polkadot-JS UI and click "Confirm Claim."



At this point you will see a success message if everything went right and your KSM will now be in the account that you claimed to. Congratulations you can now participate in aspects of the Kusama network such as governance and staking. During the soft launch period balance transfers will not be enabled.



]

Verifying your Claim

After you make an on-chain claim for KSM, your balance should be updated on Polkadot-JS Apps immediately.

Having trouble? Send an email to support@polkadot.network.

Third Party Claims Processes

We do not recommend using a third-party app or process to perform your claim or acquire KSM.

Claiming using a third-party process can lead to the loss of your allocation, therefore we cannot recommend using any third party apps to do so. Manually specifying your transaction data, as specified in our claims process, is the only way to be certain you will receive your allocation.

Kusama Endpoints

When interacting with the Kusama network via Polkadot-JS Apps or other UIs and programmatic methods, you'd ideally be running your own node (text guide, video guide). Granted, that's not something everyone wants to do, so convenience trumps ideals in most cases. To facilitate this convenience, Kusama has several public endpoints you can use for your apps.

Parity Archive Node

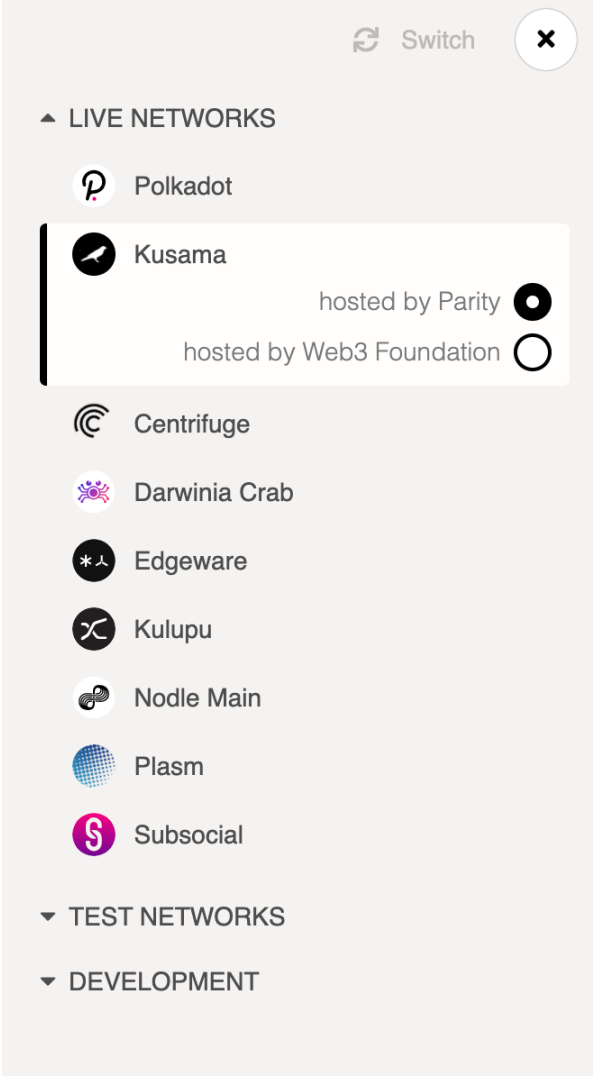
Parity, the company that develops the Polkadot Rust client, maintains an archive node at endpoint `wss://kusama-rpc.polkadot.io/`.

To connect to the Parity node, use the endpoint in your JavaScript apps like so:

```
const{ ApiPromise, WsProvider } = require('@polkadot/api')

(async () => {
    const provider = new WsProvider('wss://kusama-rpc.polkadot.io/')
    const api = await ApiPromise.create({ provider })
    // ...
```

or in Polkadot-JS Apps by clicking the top-left corner of the screen and selecting the appropriate option:



Web3 Foundation Archive Node

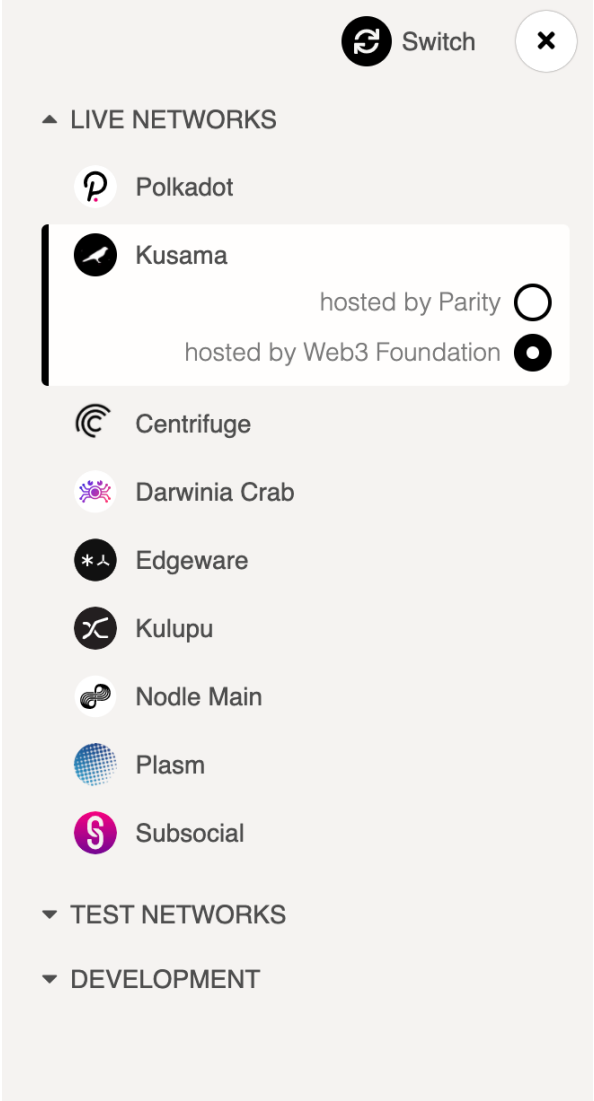
The Web3 Foundation maintains an archive node at endpoint `wss://cc3-5.kusama.network/`.

To connect to this node, use the endpoint in your JavaScript apps like so:

```
const{ ApiPromise, WsProvider } = require('@polkadot/api')

(async () => {
    const provider = new WsProvider('wss://cc3-5.kusama.network/')
    const api = await ApiPromise.create({ provider })
    // ...
```

Connect to it in Polkadot Apps UI by clicking the top-left corner of the screen and selecting the appropriate option:


Switch
LIVE NETWORKS
Polkadot
Kusama
hosted by Parity
hosted by Web3 Foundation
Centrifuge
Darwinia Crab
Edgeware
Kulupu
Nodle Main
Plasm
Subsocial
TEST NETWORKS
DEVELOPMENT

Balance Transfers

Balance transfers are used to send balance from one account to another account. To start transferring balances, we will begin by using Polkadot-JS Apps. This guide assumes that you've already created an account and have some funds that will be transferred.

Polkadot-JS Apps

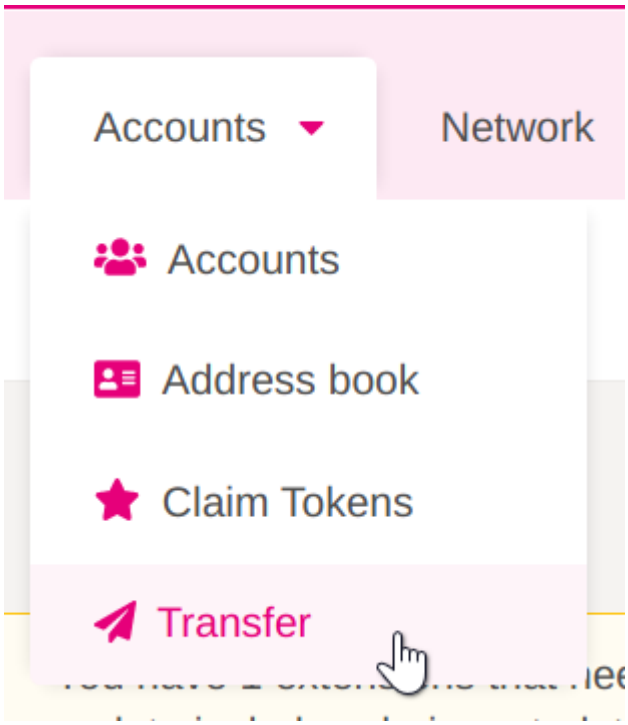
> NOTE: In this walkthrough we will be using the Polkadot network, but this is the same process for Kusama. If you would like to switch to a different network, you can change it by clicking the top left navigation dropdown and selecting a different network.

Let's begin by opening Polkadot-JS Apps. There are two ways to make a balance transfer:

1. By using the "Transfer" tab in the "Accounts" dropdown (located on the top navigational menu).
2. Clicking the "send" button while in the "Accounts" page.

Using the Transfer Tab

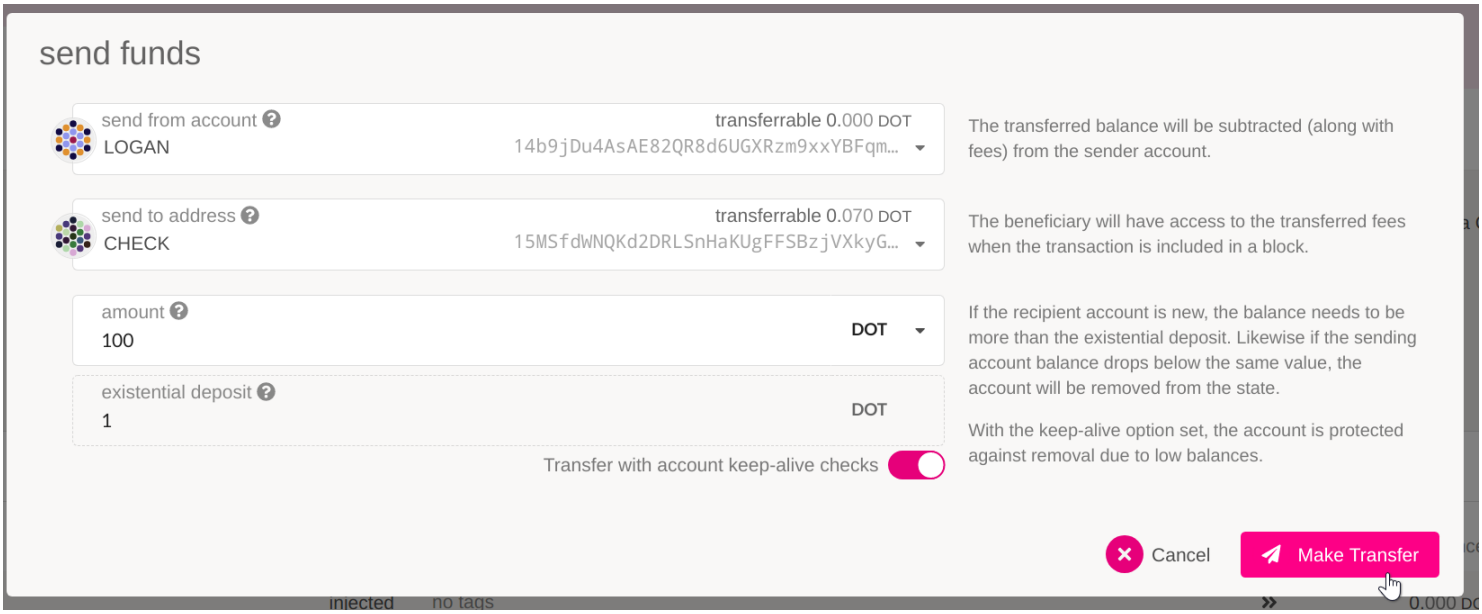
Click on the "Transfer" tab in the "Accounts" dropdown.



Now a modal window will appear on the page. The modal asks you to enter 3 inputs:

- "send from account": Your account with funds that you will send from.
- "send to address": The address of the account that will receive the funds.
- "amount": The amount of tokens you will transfer.

The "existential deposit" box shows you the minimum amount of funds you must keep in the account for it to remain active. See the existential deposit section for more information.



After setting your inputs correctly, click the "Make Transfer" button and confirm. Once the transfer is included in a block you will see a green notification in the top-right corner of your screen.

Keep-Alive Checks

At an extrinsic level, there are two main ways to transfer funds from one account to another. These are `transfer` and `transfer_keep_alive`. `transfer` will allow you to send DOTs regardless of the consequence; `transfer_keep_alive` will not allow you to send an amount that would allow the sending account to be removed due to it going below the existential deposit.

By default, Polkadot-JS Apps will use `transfer_keep_alive`, ensuring that the account you send from cannot drop below the existential deposit (1 DOT or 0.001666 KSM). However, it may be that you do not want to keep this account alive (for example, because you are moving all of your funds to a different address). In this case, click on the "keep-alive" toggle at the bottom of the modal window. The label should switch from "Transfer with account keep-alive checks" (`transfer_keep_alive` will be used) to "Normal transfer without keep-alive checks" (`transfer` extrinsic will be used). As a common use case for using normal transfers is to entirely clear out the account, a second toggle will appear if you have the keep-alive check turned off that will send all the tokens in the account, minus a transaction fee, to the destination address.

Note that attempting to send less than the existential deposit to an account with 0 DOT will always fail, no matter if the keep-alive check is on or not. For instance, attempting to transfer 0.1 DOT to an

account you just generated (and thus has no DOT) will fail, since 0.1 is less than the existential deposit of 1 DOT and the account cannot be initialized with such a low balance.

Note that even if the transfer fails due to a keep-alive check, the transaction fee will be deducted from the sending account if you attempt to transfer.

Existing Reference Error

If you are trying to reap an account and you recieve an error similar to "There is an existing reference count on the sender account. As such the account cannot be reaped from the state", then you have existing references to this account that must first be removed before it can be reaped. References may still exist from:

- Bonded tokens (most likely)
- Unpurged session keys (if you were prevously a validator)
- Token locks
- Existing recovery info
- Existing assets

Bonded Tokens

If you have tokens that are bonded, you will need to unbond them before you can reap your account. Follow the instructions at Unbonding and Rebonding to check if you have bonded tokens, stop nominating (if necessary) and unbond your tokens.

Purging Session Keys

If you used this account to set up a validator and you did not purge your keys before unbonding your tokens, you need to purge your keys. You can do this by seeing the How to Stop Validating page. This can also be checked by checking `session.nextKeys` in the chain state for an existing key.

Checking for Locks

You can check for locks by querying `system.account(AccountId)` under `Developer > Chain state`. Select your account, then click the "+" button next to the dropdowns, and check the relative `data` JSON object. If you see a non-zero value for anything other than `free`, you have locks on your account that need to get resolved.

You can also check for locks by navigating to `Accounts > Accounts` in PolkadotJS Apps. Then, click the dropdown arrow of the relevant account under the 'balances' column. If it shows that some tokens are in a 'locked' state, you can see why by hovering over the information icon next to it.

Existing Recovery Info

{{ polkadot: Currently, {{ polkadot: Polkadot :polkadot }} {{ kusama: Kusama :kusama }} does not use the Recovery Pallet, so this is probably not why your tokens have existing references. :polkadot}}

{{ kusama: On Kusama, you can check if recovery has been set up by checking the `recovery.recoverable(AccountId)` chain state. This can be found under `Developer > Chain state` in PolkadotJS Apps. :kusama}}

Existing {{ polkadot: Non-DOT :polkadot}} {{ kusama: Non-KSM :Kusama }} Assets

Currently, {{ polkadot: Polkadot :polkadot }} {{ kusama: Kusama :kusama }} does not use the Assets Pallet, so this is probably not why your tokens have existing references.

From the Accounts Page

Navigate to the "Accounts" page by selecting the "Accounts" tab from the "Accounts" dropdown located on the top navigational menu of Polkadot-JS Apps.

You will see a list of accounts you have loaded. Click the "Send" button in the row for the account you will like to send funds from.

Now you will see the same modal window as if using the "Transfer" tab. Fill in the inputs correctly and hit "Make Transfer" then confirm the balance transfer. You will see a green notification in the top-right corner of the screen when the transfer is included in a block.

Thousand Validators Programme

The Thousand Validators Programme is an initiative by Web3 Foundation and Parity Technologies to use the funds held by both organizations to nominate validators in the community.

It serves two major purposes: 1) to give validators a structured on-ramp to join the active set of validators on Kusama and Polkadot and 2) to further decentralize the validator active set.

How it Works

The nominating backend will routinely change its nominations at every era (or every 4 eras on Kusama). It will nominate validators which fit all the requirements and are eligible. Of this pool, it will nominate as many as possible although some validators which are eligible might not receive nominations every round (due to the constraint of nominating a maximum of 16 validators per nominator). If a validator is active during a single nomination period (the time after a new nomination and before the next one) and does not break any of the requirements, it will have its rank increased by 1. Validators with higher rank have performed well within the programme for a longer period of time.

Setting up a Validator

Please see the wiki page for setting up a validator as well as additional information on making your validator secure.

How to Apply

Kusama

In order to apply to the Kusama programme, set up your node to adhere to the requirements below and fill in the application form. You will hear back from the team shortly.

Requirements

- Verified identity (see here for instructions)
- Connect to dedicated telemetry (use `--telemetry-url 'wss://telemetry-backend.w3f.community/submit 1'` when starting the node)

- Minimum of 50 KSM self-stake
- No more than 10% commission
- Separate controller and stash (or have a Staking proxy set up)
- Must be on the latest release
- Max two nodes (under same sub/super identity)

Leaderboard

The leaderboard is available at https://thousand-validators.kusama.network/#/leaderboard.

Nominators

The below addresses are the stash / controller pairs for the nominators involved in the Kusama Thousand Validators programme. They are formatted like "`stash` / `controller`".

- `G1rrUNQSk7CjjEmLSGcpNu72tVtyzbWdUvgmSer9eBitXWf` / `H9BFvNPTqDEmWZ63M82ohrFmvEFASm25ErUMzmXDrbAr1kq`
- `HgTtJusFEn2gmMmB5wmJDnMRXKD6dzqCpNR7a99kkQ7BNvX` / `H4UgNEEN92YXz96AyQgwkJQSpXGdptYLkj9jXVKrNXjQHRJ`
- `EX9uchmfeSqKTM7cMMg8DkH49XV8i4R7a7rqCn8btpZBHDP` / `H54GA3nq3xeNrdbHkepAufSPMjaCxxkmfej4PosqD84bY3V`
- `H4635Bjj3X7TjnQhd55p9DyFPK39JiRypmCnsDhS3NHSMS5` / `CeB8SLnJivXRtC5PgXchrece8j3TBQRaqfGqHngvhD3LRHD`
- `Hs94zeHrSUWG1VBzsvHPxR2VRN5mq1Rib1PEfjH7wkGzv2Z` / `HCbe2ZFujLNYsrKGub8XKGJuky3LTAF6NhWnRTs6NyDaVkQ`
- `CdWjVn5J9ct4D3yK8HbwXmmGyLcxjkDitaBLxwH5g5Vh7pi` / `Dicn4AxJRsnJ6sRsYPNZvC3xrRhvNBanfPP79haVL7ywyPn`
- `HxRmQTVrMxMkhyZquYLu2hSL1QDYvVwSpDfBHvVJhEMVzRj` / `CbFFE91fYzkKsuFjSjfQrc7Bz2bbM9vYcQgzijxHd4LtoKw`
- `FJbKWFGCfZFZiNQtAZ5uqbUhKeB3P3a6RVRw6GqTbgiqtem` / `Gt2p2gZHPvHCvPUuT2BKaeAAADPgVEa9eXMQBn74RwMa6mX`

A time delay proxy is used as the interaction method for some of these accounts.

Since approximately early January 2021, the nominators will select an automatic number of validators to nominate based on the lowest amount staked for a validator and the amount of funds it holds. This can be anywhere from a few validators receiving nomination from a single nominator to the max of 16.

Polkadot

> Note: Entrance to the Polkadot programme requires a rank of 25 or higher in the Kusama programme. This usually takes about a month.

In order to apply to the Polkadot programme, set up your node to adhere to the requirements below and fill in the application form. You will hear back from the team shortly.

Requirements

- Verified identity (see here for instructions)
- Connect to dedicated Telemetry (use `--telemetry-url 'wss://telemetry-backend.w3f.community/submit 1'` when starting the node)
- Rank 25 or higher on Kusama Thousand Validators Programme
- Minimum of 5_000 DOTs self stake (exceptions by approval for good intentions)
- Reward destination 'Staked'
- No more than 3% commission
- Separate stash and controller (or have a Staking proxy set up)
- Must be on the latest release

Nominators

The below addresses are the stash / controller pairs for the nominators involved in the Polkadot Thousand Validators programme. They are formatted like "`stash` / `controller`".

- `14Ns6kKbCoka3MS4Hn6b7oRw9fFejG8RH5rq5j63cWUfpPDJ` / `16XJHQ58dEPnZn5J5YqmRcJmKtvVFFMoMrXgj6fWJfeGGkQw`
- `12RYJb5gG4hfoWPK3owEYtmWoko8G6zwYpvDYTyXFVSfJr8Y` / `13GLXK1TZKKDM9aRBBK3VYZymHjKChtQjJznsRqaR9dwwrQU`
- `16GMHo9HZv8CcJy4WLoMaU9qusgzx2wxKDLbXStEBvt5274B` / `16eM1npMwKzpGy48NDna1jC6P71S783wjpbdeKT8RgzQx8Jd`

A time delay proxy is used as the main interaction method for all of these accounts.

Since approximately early January 2021, the nominators will select an automatic number of validators to nominate based on the lowest amount staked for a validator and the amount of funds it holds. This can be anywhere from a few validators receiving nominations from a single nominator to the max of 16.

Parachain Slots Auction

For a parachain to be added to Polkadot it must inhabit one of the available parachain slots. A parachain slot is a scarce resource on Polkadot and only a limited number will be available. As parachains ramp up there may only be a few slots that are unlocked every few months. The goal is to eventually have 100 parachain slots available on {{ polkadot: Polkadot :polkadot }} {{ kusama: Kusama :kusama }} (these will be split between parachains and the parathread pool). If a parachain wants to have guaranteed block inclusion at every Relay Chain block, it must acquire a parachain slot.

The parachain slots will be sold according to an unpermissioned candle auction that has been slightly modified to be secure on a blockchain.

Mechanics of a Candle auction

Candle auctions are a variant of open auctions where bidders submit bids that are increasingly higher and the highest bidder at the conclusion of the auction is considered the winner.

Candle auctions were originally employed in 16th century for the sale of ships and get their name from the "inch of a candle" that determined the open period of the auction. When the flame extinguished and the candle went out, the auction would suddenly terminate and the standing bid at that point would win.

When candle auctions are used online, they require a random number to decide the moment of termination.

Parachain slot auctions will differ slightly from a normal candle auction in that it does not use the random number to decide the duration of its opening phase. Instead, it has a known open phase and will be retroactively determined (at the normal close) to have ended at some point in the past during the ending phase. So during the open phase, bids will continue to be accepted, but later bids have higher probability of losing since the retroactively determined close moment may be found to have preceded the time that a bid was submitted.

Rationale

The open and transparent nature of blockchain systems opens attack vectors that are non-existent in traditional auction formats. Normal open auctions in particular can be vulnerable to *auction sniping* when implemented over the internet or on a blockchain.

Auction sniping takes place when the end of an auction is known and bidders are hesitant to bid their true price early, in hopes of paying less than they actually value the item.

For example, Alice may value an item at auction for 30 USD. She submits an initial bid of 10 USD in hopes of acquiring the items at a lower price. Alice's strategy is to place incrementally higher bids until her true value of 30 USD is exceeded. Another bidder Eve values the same item at 11 USD. Eve's strategy is to watch the auction and submit a bid of 11 USD at the last second. Alice will have no time to respond to this bid before the close of the auction and will lose the item. The auction mechanism is sub-optimal because it has not discovered the true price of the item and the item has not gone to the actor who valued it the most.

On blockchains this problem may be even worse, since it potentially gives the producer of the block an opportunity to snipe any auction at the last concluding block by adding it themselves and/or ignoring other bids. There is also the possibility of a malicious bidder or a block producer trying to *grief* honest bidders by sniping auctions.

For this reason, Vickrey auctions, a variant of second price auction in which bids are hidden and only revealed in a later phase, have emerged as a well-regarded mechanic. For example, it is implemented as the mechanism to auction human readable names on the ENS. The Candle auction is another solution that does not need the two-step commit and reveal schemes (a main component of Vickrey auctions), and for this reason allows smart contracts to participate.

Candle auctions allow everyone to always know the states of the bid, but not when the auction will be determined to have ended. This helps to ensure that bidders are willing to bid their true bids early. Otherwise, they might find themselves in the situation that the auction was determined to have ended before they even bid.

Polkadot Implementation

Polkadot will use a *random beacon* based on the VRF that's used also in other places of the protocol. The VRF will provide the base of the randomness, which will retroactively determine the end-time of the auction.

The slot durations are capped to {{ polkadot: 2 years and divided into 3-month periods :polkadot }} {{ kusama: 1 year and divided into 6-week periods :kusama }}; Parachains may lease a slot for any combination of periods of the slot duration. Parachains may lease more than one slot over time, meaning that they could extend their lease to Polkadot past the maximum duration by leasing a contiguous slot.

> Note: Individual parachain slots are fungible. This means that parachains do not need to always inhabit the same slot, but as long as a parachain inhabits any slot it can continue as a

> parachain.

Bidding

Parachains, or parachain teams, can bid in the auction by specifying the slot range that they want to lease as well as the number of tokens they are willing to reserve. Bidders can be either ordinary accounts, or use the crowdloan functionality to source tokens from the community.

```
Parachain slots at genesis

       --3 months--
       v          v
Slot A |     1    |     2     |     3     |     4     |     5     |     6    |
7     |     8     |     9     |...
Slot B |     1    |     2     |     3     |     4     |     5     |     6    |
7     |     8     |     9     |...
Slot C |__________|     1     |     2     |     3     |     4     |     5    |
6     |     7     |     8     |     9     |...
Slot D |__________|     1     |     2     |     3     |     4     |     5    |
6     |     7     |     8     |     9     |...
Slot E |__________|__________|     1     |     2     |     3     |     4    |
5     |     6     |     7     |     8     |     9     |...
       ^
^
       ---------------------------------------------max lease-------------------
--------------------
```

Each period of the range 1 - 4 represents a {{ polkadot: 3-month duration for a total of 2 years :polkadot }} {{ kusama: 6-week duration for a total of 1 year :kusama }}

Bidders will submit a configuration of bids specifying the token amount they are willing to bond and for which periods. The slot ranges may be any of the periods 1 - `n`, where `n` is the number of periods available for a slot (`n` will be 8 for both Polkadot and Kusama).

> Please note: If you bond tokens with a parachain slot, you cannot stake with those tokens. In this way, you pay for the parachain slot by forfeiting the opportunity to earn staking rewards.

A bidder configuration for a single bidder may look like the following pseudocode example:

```
const bids = [
  {
    range: [1, 2, 3, 4, 5, 6, 7, 8],
```

```
    bond_amount: 300,
  },
  {
    range: [1, 2, 3, 4],
    bond_amount: 777,
  },
  {
    range: [2, 3, 4, 5, 6, 7],
    bond_amount: 450,
  },
];
```

The important concept to understand from this example is that bidders may submit different configurations at different prices (`bond_amount`). However, only one of these bids would be eligible to win exclusive of the others.

The winner selection algorithm will pick bids that may be non-overlapping in order to maximize the amount of tokens held over the entire lease duration of the parachain slot. This means that the highest bidder for any given slot lease period might not always win (see the example below).

A random number, which is based on the VRF used by Polkadot, is determined at each block. Additionally, each auction will have a threshold that starts at 0 and increases to 1. The random number produced by the VRF is examined next to the threshold to determine if that block is the end of the auction withing the so-called *ending period*. Additionally, the VRF will pick a block from the last epoch to take the state of bids from (to mitigate some types of attacks from malicious validators).

Examples

There is one parachain slot available.

Charlie bids `75` for the range 1 - 8.

Dave bids `100` for the range 5 - 8.

Emily bids `40` for the range 1 - 4.

Let's calculate each bidder's valuation according to the algorithm. We do this by multiplying the bond amount by the number of periods in the specified range of the bid.

Charlie - 75 * 8 = 600 for range 1 - 8

Dave - 100 * 4 = 400 for range 5 - 8

Emily - 40 * 4 = 160 for range 1 - 4

Although Dave had the highest bid in accordance to token amount, when we do the calculations we see that since he only bid for a range of 4, he would need to share the slot with Emily who bid much less. Together Dave's and Emily's bids only equal a valuation of `560`.

Charlie's valuation for the entire range is `600` therefore Charlie is awarded the complete range of the parachain slot.

FAQ

Why doesn't everyone bid for the max length?

For the duration of the slot the tokens bid in the auction will be locked up. This means that there are opportunity costs from the possibility of using those tokens for something else. For parachains that are beneficial to Polkadot, this should align the interests between parachains and the Polkadot Relay Chain.

How does this mechanism help ensure parachain diversity?

The method for dividing the parachain slots into intervals was partly inspired by the desire to allow for a greater amount of parachain diversity, and prevent particularly large and well-funded parachains from hoarding slots. By making each period a {{ polkadot: three-month duration but the overall slot a 2-year duration :polkadot }} {{ kusama: 6-week duration but the overall slot a 1-year duration :kusama }}, the mechanism can cope with well-funded parachains that will ensure they secure a slot at the end of their lease, while gradually allowing other parachains to enter the ecosystem to occupy the durations that are not filled. For example, if a large, well-funded parachain has already acquired a slot for range 1 - 8, they would be very interested in getting the next slot that would open for 2 - 9. Under this mechanism that parachain could acquire period 5 (since that is the only one it needs) and allow range 2 - 8 of the second parachain slot to be occupied by another.

Why is randomness difficult on blockchains?

Randomness is problematic for blockchain systems. Generating a random number trustlessly on a transparent and open network in which other parties must be able to verify opens the possibility for actors to attempt to alter or manipulate the randomness. There have been a few solutions that have been put forward, including hash-onions like RANDAO and verifiable random functions (VRFs). The latter is what Polkadot uses as a base for its randomness.

Are there other ways of acquiring a slot besides the candle auction?

Another way, besides the candle auction, to acquire a parachain slot is through a secondary market where an actor who has already won a parachain slot can resell the slot along with the associated deposit of tokens that is locked up to another buyer. This would allow the seller to get liquid tokens in exchange for the parachain slot and the buyer to acquire the slot as well as the deposited tokens.

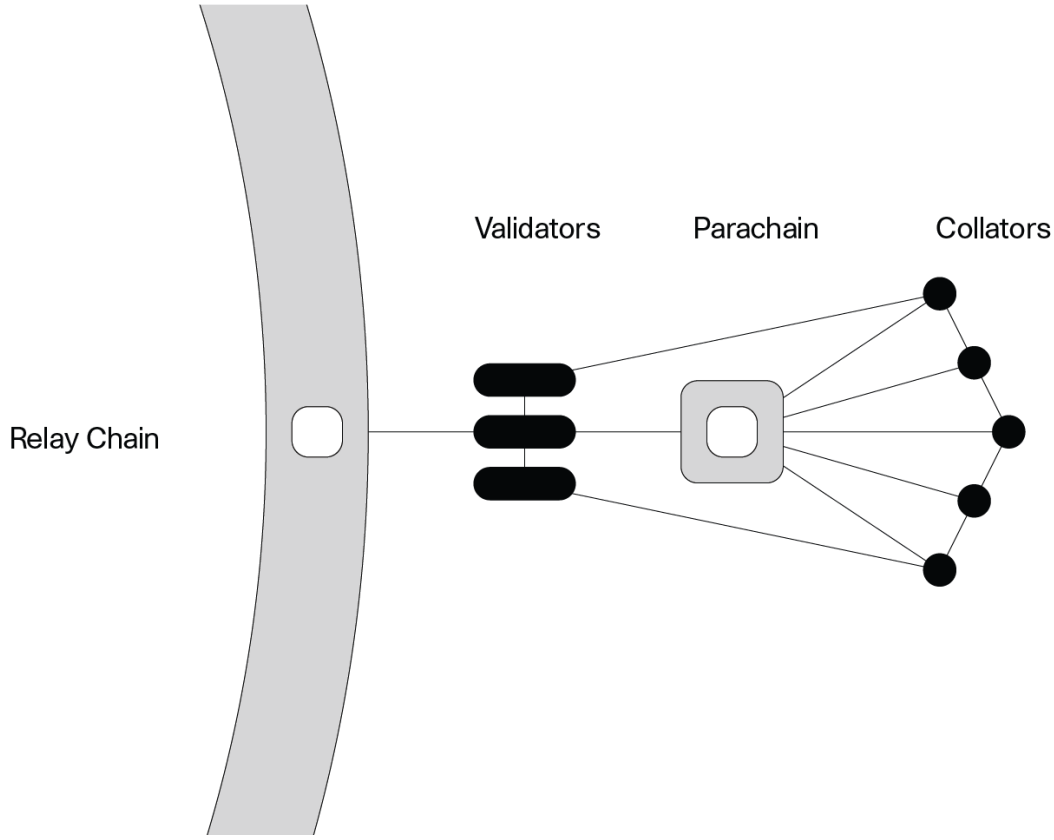
A number of system-level parachains may be granted slots by the governing bodies of the Relay Chain. Such parachains would not have to bid for or renew their slots as they would be considered essential to the ecosystem's future.

Resources

- Parachain Allocation - W3F research page on parachain allocation that goes more in depth to the mechanism.

Parachains

> Note: For information on how to participate in the crowdloan and parachain auction testing on Rococo, please see the {{ polkadot: Rococo page :polkadot }} > {{ kusama: Rococo page :kusama }}.



A parachain is an application-specific data structure that is globally coherent and validatable by the validators of the Relay Chain. Most commonly a parachain will take the form of a blockchain, but there is no specific need for them to be actual blockchains. They take their name from the concept of parallelized chains that run parallel to the Relay Chain. Due to their parallel nature, they are able to parallelize transaction processing and achieve scalability of the {{ polkadot: Polkadot :polkadot }} {{ kusama: Kusama :kusama }} system. They share in the security of the entire network and can communicate with other parachains through XCMP.

Parachains are maintained by a network maintainer known as a collator. The role of the collator node is to maintain a full-node of the parachain, retain all necessary information of the parachain, and produce new block candidates to pass to the Relay Chain validators for verification and inclusion in the shared state of Polkadot. The incentivization of a collator node is an implementation detail of the parachain. They are not required to be staked on the Relay Chain or own DOT tokens unless stipulated to do so by the parachain implementation.

The Polkadot Host (PH) requires that the state transitions performed on parachains to be specified as a Wasm executable. Proofs of new state transitions that occur on a parachain must be validated against the registered state transition function (STF) that is stored on the Relay Chain by the validators before {{ polkadot: Polkadot :polkadot }} {{ kusama: Kusama :kusama }} acknowledges a state transition has occurred on a parachain. The key constraint regarding the logic of a parachain is that it must be verifiable by the Relay Chain validators. Verification most commonly takes the form of a bundled proof of a state transition known as a Proof-of-Verification (PoV) block, which is submitted to the validators from one or more of the parachain collators to be checked.

Parachain Economies

Parachains may have their own economies with their own native tokens. Schemes such as Proof-of-Stake are usually used to select the validator set in order to handle validation and finalization; parachains will not be required to do either of those things. However, since {{ polkadot: Polkadot :polkadot }} {{ kusama: Kusama :kusama }} is not overly particular about what the parachain can implement, it may be the choice of the parachain to implement a staking token, but it's not generally necessary.

Collators may be incentivized through inflation of a native parachain token. There may be other ways to incentivize the collator nodes that do not involve inflating the native parachain token.

Transaction fees in a native parachain token can also be an implementation choice of parachains. {{ polkadot: Polkadot :polkadot }} {{ kusama: Kusama :kusama }} makes no hard and fast rules for how the parachains decide on original validity of transactions. For example, a parachain may be implemented so that transactions must pay a minimum fee to collators to be valid. The Relay Chain will enforce this validity. Similarly, a parachain could not include that in their implementation and Polkadot would still enforce its validity.

Parachains are not required to have their own token. If they do, is up to the parachain to make the economic case for their token, not {{ polkadot: Polkadot :polkadot }} {{ kusama: Kusama :kusama }}.

Parachain Hubs

While Polkadot enables crosschain functionality amongst the parachains, it necessitates that there is some latency between the dispatch of a message from one parachain until the destination parachain receives the message. In the optimistic scenario, the latency for this message should be at least 2 blocks - one block for the message to be dispatched and one block for the receiving parachain to process and produce a block that acts upon the message. However, in some cases we may see that the latency for messages is higher if there are many messages that are in queue to be

processed, or if there exist no nodes that are running both of the parachain networks that can quickly gossip the message across the networks.

Due to the neccesary latency involved in sending crosschain messages, some parachains are planning to become *hubs* for an entire industry. For example, a parachain project Acala is planning to become a hub for decentralized finance (DeFi) applications. Many DeFi applications take advantage of a property known as *composability* which means that functions of one application can be composed with others in a synergistic way to create new applications. One example of this include flash loans, which borrow funds to execute some on-chain logic as long as the loan is repaid at the end of the transaction.

An issue with crosschain latency means that the property of composability is weakened among parachains compared to a single blockchain. **This implication is common to all sharded blockchain designs, including Polkadot, Eth2.0, and others.** The solution to this is the introduction of parachain hubs which maintain the stronger property of single block composability.

Parachain Slot Acquisition

{{ polkadot: Polkadot :polkadot }} {{ kusama: Kusama :kusama }} supports a limited number of parachains, currently estimated to be about 100. As the number of slots is limited, there are several ways to allocate them:

- Governance granted parachains, or "common good" parachains
- Auction granted parachains
- Parathreads

"Common Good" parachains are allocated by Polkadot's on-chain {{ polkadot: governance :polkadot }} {{ kusama: governance :kusama }} system, and are deemed as a "common good" for the network, such as bridges to other networks or chains. They are usually considered system level chains or public utility chains. These typically do not have an economic model of their own and help remove transactions from the Relay Chain, allowing for more efficient parachain processing.

{{ polkadot: Auction granted parachains :polkadot }} {{ kusama: Auction granted parachains :kusama }} are granted in a permissionless auction. Parachain teams can either bid with their own DOT tokens, or source them from the community using the {{ polkadot: crowdloan functionality :polkadot }} {{ kusama: crowdloan functionality :kusama }}.

{{ polkadot: Parathreads :polkadot }} {{ kusama: Parathreads :kusama }} have the same API as parachains, but are scheduled for execution on a pay-as-you-go basis with an auction for each block.

Slot Expiration

When a parachain wins an auction, the tokens that it bid gets reserved until the end of the lease. Reserved balances are non-transferrable and cannot be used for staking. At the end of the lease, the tokens are unreserved. Parachains that have not secured a new lease to extend their slot will automatically become parathreads.

Common Good Parachains

"Common Good" parachains are parachain slots reserved for functionality that benefits the the ecosystem as a whole. By allocating a subset of parachain slots to common good chains, the entire network can realize the benefit of valuable parachains that would otherwise be underfunded due to the free-rider problem. They are not allocated via the parachain auction process, but by the on-chain {{ polkadot: governance :polkadot }} {{ kusama: governance :kusama }} system.

The purpose of these parachains will probably fall into one of two categories: system level chains or public utility chains.

System Level Chains

System level chains move functionality from the Relay Chain into parachains, minimizing the administrative use of the Relay Chain. For example, a governance parachain could move all the governance processes from the Relay Chain into a parachain. Adding a system level chain is generally uncontroversial, because they merely move functionality that the stakeholders already agreed was useful from one place (the Relay Chain) to another (a parachain).

Moving the logic from the Relay Chain to a parachain is an optimization that makes the entire network more efficient. All validators need to process all Relay Chain transactions, but split into small groups to validate parachains in parallel. By moving system level logic to a parachain, and allowing the processing to be done by a subgroup of validators instead of all, it frees capacity in the Relay Chain for its primary function: validating parachains. Adding a system level chain could make the network capable of processing several more parachains. Rather than taking a slice of a 100 parachain pie, a system level chain takes one slice and bakes a bigger pie.

Examples of potential system level chains include parachains for balances, elections (for both staking and Council), governance, and identity. Eventually, the Relay Chain could become transactionless, as in, it would only validate parachain state transitions and all of its current transactional functionality would exist within parachains.

The vast majority of common good chains will likely be the unopinionated system level chains.

Public Utility Chains

Public utility chains add functionality that doesn't exist yet, but that the stakeholders believe will add value to the entire network. Because public utility chains add new functionality, there is a subjective component to their addition: the stakeholders of the network must believe that it is worth allocating a slot that would otherwise go to the winners of an auction, and thus would have an objective expression of conviction from its backers. Governance provides the means to internalize the value of the parachain slot and distribute it across all members of the network.

Public utility chains will always be fully aligned with their Relay Chain stakeholder base. This means that they will adopt the Relay Chain's native token (i.e. DOT or KSM) as their native token and respect any messages incoming from the Relay Chain and system level parachains at face value.

Some examples of potential public utility chains are bridges, DOT/KSM-denominated smart contract platforms, and generic asset chains. All of these could operate without a new token:

- A bridge could add its own native token to charge as a toll, but in many cases that would be arbitrary value capture, when it could just as well use DOT/KSM and/or the bridged chain's assets in its fee mechanism.
- A DOT/KSM-denominated smart contract layer-one blockchain would allow Wasm smart contract execution using DOT/KSM as the native asset with which to pay gas.
- A generic assets chain would allow anyone to place a deposit in DOT/KSM to deploy their asset on-chain. Assets on this chain could be backed by physical goods like artwork, real estate, or gold; or by paper goods like shares in a company or fiat currency held by a trusted party, providing a stable, permanent launchpad for stablecoins and Central Bank Digital Currencies.

Public utility parachains would typically grant privileged business logic to Polkadot's governance. Just as the Polkadot Relay Chain has several privileged functions like setting the validator count or allocating DOT from the Treasury, these parachains can have privileged functions like changing system parameters or registering an asset.

Because public utility chains add functionality beyond the scope of the Relay Chain, they will likely be approved by the network stakeholders only in rare scenarios.

Examples

Some examples of parachains:

- Encrypted Consortium Chains: These are possibly private chains that do not leak any information to the public, but still can be interacted with trustlessly due to the nature of the XCMP protocol.

- **High Frequency Chains**: These are chains that can compute many transactions in a short amount of time by taking certain trade-offs or making optimizations.
- **Privacy Chains**: These are chains that do not leak any information to the public through use of novel cryptography.
- **Smart Contract Chains**: These are chains that can have additional logic implemented on them through the deployment of code known as *smart contracts*.

FAQ

What is "parachain consensus"?

"Parachain consensus" is special in that it will follow the Polkadot Relay Chain. Parachains cannot use other consensus algorithms that provide their own finality. Only sovereign chains (that must bridge to the Relay Chain via a parachain) can control their own consensus. Parachains have control over how blocks are authored and by whom.

How will parachain slots be distributed?

Parachain slots will be acquirable through auction, please see the {{ polkadot: parachain slots :polkadot }} {{ kusama: parachain slots :kusama }} article. Additionally, some parachain slots will be set aside to run {{ polkadot: parathreads :polkadot }} {{ kusama: parathreads :kusama }} — chains that bid on a per-block basis to be included in the Relay Chain.

What happens to parachains when the number of validators drops below a certain threshold?

The minimal safe ratio of validators per parachain is 5:1. With a sufficiently large set of validators, the randomness of their distribution along with availability and validity will make sure security is on-par. However, should there be a big outage of a popular cloud provider or another network connectivity catastrophe, it is reasonable to expect that the number of validators per chain will drop.

Depending on how many validators went offline, the outcome differs.

If a few validators went offline, the parachains whose validator groups will be too small to validate a block will skip those blocks. Their block production speed will slow down to any increment of 6 seconds, until the situation is resolved and the optimal number of validators is in that parachain's validator group again.

If anywhere from 30% to 50% of the validators go offline, availability will suffer because we need two thirds of the validator set to back the parachain candidates. In other words, all parachains will

stop until the situation is resolved. Finality will also stop, but low-value transactions on the relay chain should be safe enough to execute, despite common forks. Once the required number of validators is in the validator set again, parachains will resume block production.

Given that collators are full nodes of the relay chain and the parachain they are running, they will be able to recognize a disruption as soon as it occurs and should stop producing block candidates. Likewise, it should be easy for them to recongize when it's safe to restart block production - perhaps based on finality delay, validator set size, or some other factor that is yet to be decided within Cumulus.

Parachain Development Kits (PDKs)

Parachain Development Kits are a set of tools that enable developers to create their own applications as parachains. For more info see here.

Deploying parachains

Please see the Cumulus repository README for information on compiling and deploying a parachain.

Resources

- Polkadot: The Parachain - Blog post by Polkadot co-founder Rob Habermeier that introduced parachains in 2017 as "a simpler form of blockchain, which attaches to the security provided by a Relay Chain rather than providing its own. The Relay Chain provides security to attached parachains, but also provides a guarantee of secure message-passing between them."
- The Path of a Parachain Block - A technical walkthrough of how parachains interact with the Relay Chain.

Parathreads

Parathreads are an idea for parachains to temporarily participate (on a block by block basis) in Polkadot security without needing to lease a dedicated parachain slot. This is done through economically sharing the scarce resource of a *parachain slot* among a number of competing resources (parathreads). Chains that otherwise would not be able to acquire a full parachain slot, or do not find it economically sensible to do so, are enabled to participate in Polkadot's shared security — albeit with an associated fee per executed block. It also offers a graceful off-ramp to parachains that no longer require a dedicated parachain slot, but would like to continue using the Relay Chain.

Origin

According to this talk in Chengdu, the origin of the idea came from similar notions in the limited resource of memory on early personal computers of the late '80s and '90s. Since computers have a limited amount of physical memory, when an application needs more, the computer can create virtual memory by using *swap space* on a hard disk. Swap space allows the capacity of a computer's memory to expand and for more processes to run concurrently with the trade-off that some processes will take longer to progress.

Parachain vs. Parathread

Parachains and parathreads are very similar from a development perspective. One can imagine that a chain developed with Substrate can at different points in its lifetime assume one of three states: an independent chain with secured bridge, a parachain, or a parathread. It can switch between these last two states with relatively minimal effort since the difference is more of an economic distinction than a technological one.

Parathreads have the exact same benefits for connecting to Polkadot that a full parachain has. Namely, it is able to send messages to other para{chain,threads} through XCMP and it is secured under the full economic security of Polkadot's validator set.

The difference between parachains and parathreads is economic. Parachains must be registered through a normal means of Polkadot, i.e. governance proposal or parachain slot auction. Parathreads have a fixed fee for registration that would realistically be much lower than the cost of acquiring a parachain slot. Similar to how DOT are locked for the duration of parachain slots and then returned to the winner of the auction, the deposit for a parathread will be returned to the parathread after the conclusion of its term.

Registration of the parathread does not guarantee anything more than the registration of the parathread code to the Polkadot Relay Chain. When a parathread progresses by producing a new block, there is a fee that must be paid in order to participate in a per-block auction for inclusion in the verification of the next Relay Chain block. All parathreads that are registered are competing in this auction for their parathread to be included for progression.

There are two interesting observations to make about parathreads. One, since they compete on a per-block basis, it is similar to how transactions are included in Bitcoin or Ethereum. A similar fee market will likely develop, which means that busier times will drive the price of parathread inclusion up, while times of low activity will require lower fees. Two, this mechanism is markedly different from the parachain mechanism, which guarantees inclusion as long as a parachain slot is held; parathread registration grants no such right to the parathread.

How Will Parathreads be Operated?

A portion of the parachain slots on the Relay Chain will be designated as part of the parathread pool. In other words, some parachain slots will have no parachain attached to them and rather will be used as a space for which the winner(s) of the block-by-block parathread fee auction can have their block candidate included.

Collators will offer a bid designated in DOT for inclusion of a parathread block candidate. The Relay Chain block author is able to select from these bids to include a parathread block. The obvious incentive is for them to accept the block candidate with the highest bid, which would bring them the most profit. The tokens from the parathread bids will likely be split 80-20, meaning that 80% goes into Polkadot treasury and 20% goes to the block author. This is the same split that applies also to transaction fees and, like many other parameters in Polkadot, can be changed through a governance mechanism.

Parathread Economics

There are two sources of compensation for collators:

1. Assuming a parathread has its own local token system, it pays the collators from the transaction fees in its local token. If the parathread does not implement a local token, or its local token has no value (e.g. it is used only for governance), then it can use DOT to incentivize collators.
2. Parathread protocol subsidy. A parathread can mint new tokens in order to provide additional incentives for the collator. Probably, the amount of local tokens to mint for the parathread would be a function of time, the more time that passes between parathread blocks that are included in the Relay Chain, the more tokens the parathread is willing to subsidize in order to be

considered for inclusion. The exact implementation of this minting process could be through local parathread inflation or via a stockpile of funds like a treasury.

Collators may be paid in local parathread currency. However, the Relay Chain transacts with the Polkadot universal currency (DOT) only. Collators must then submit block candidates with an associated bid in DOT.

Parachain Slot Swaps

It will be possible for a parachain that holds a parachain slot to swap this slot with a parathread so that the parathread "upgrades" to a full parachain and the parachain becomes a parathread. The chain can also stop being a chain and continue as a thread without swapping the slot. The slot, if unoccupied, would be auctioned off in the next auction period.

This provides a graceful off-ramp for parachains that have reached the end of their lease and do not have sufficient usage to justify renewal; they can remain registered on the Relay Chain but only produce new blocks when they need to.

Parathreads help ease the sharp stop of the parachain slot term by allowing parachains that are still doing something useful to produce blocks, even if it is no longer economically viable to rent a parachain slot.

Resources

- Parathreads: Pay-as-you-go Parachains

Parachain Crowdloans

{{ polkadot: Polkadot :polkadot }} {{ kusama: Kusama :kusama }} allows parachains to source tokens for their parachain bids in a decentralized crowdloan.

> Note: For information on how to participate in the crowdloan and parachain auction testing on Rococo, please see the {{ polkadot: Rococo page :polkadot }} > {{ kusama: Rococo page :kusama }}.

Anyone can create a new crowdloan campaign for a parachain slot {{ kusama: by depositing a specified number of tokens :kusama }}. A campaign is configured as a range of slots (i.e. the duration of the {{ polkadot: parachain :polkadot }} {{ kusama: parachain :kusama }} will bid for), a cap, and a duration. The duration can last over several auctions, meaning that the team will not need to restart the campaign just because they do not secure a slot on their first attempt.

Each created campaign will have an index. Once a crowdloan campaign is open, anyone can participate by sending a special transaction that references the campaign's index. Tokens used to participate must be transferable — that is, not locked for any reason, including staking, vesting, and governance — because they will be moved into a module-controlled account that was generated uniquely for this campaign.

> Important: All crowdloan contributions are handled by the Crowdloan module's logic where a campaign is identified by index, not by address. **Never transfer tokens to an address in support of a campaign.**

It is up to individual parachain teams to decide if and how they want to reward participants who forgo staking and choose to lock their tokens in support of the parachain's campaign. As one can imagine, rewards will take many forms and may vary widely among projects.

During some point of the crowdloan campaign the owner will upload the parachain data. Ideally, the owner does this before soliciting contributions to the campaign so that the contributors can verify it. The data can only be uploaded once during the course of the campaign and it will be what is deployed as the parachain's runtime. Of course, once the parachain is running it can always change via runtime upgrades (as determined through its own local governance).

If a crowdloan campaign is successful, that parachain will be on-boarded to the Relay Chain. The collective tokens will be locked in that parachain's account for the entire duration that it is active.

Participants will be able to reclaim their tokens in one of two ways:

- If the campaign was successful, then the parachain will enter a retirement phase at the end of its lease. During this phase, participants can withdraw the tokens with which they participated.
- If the campaign was not successful, then this retirement phase will begin at the campaign's configured end, and participants can likewise withdraw their tokens.

Note: When the lease periods won by the crowdloan have finished, or the crowdloan has ended without winning a slot, anyone can trigger the refund of crowdloan contributions back to their original owners. All contributions must be returned before the crowdloan is fully deleted from the system.

Several teams in the ecosystem are preparing dashboards to track auctions and crowdloan campaigns. Check back with this page for updates as these are released.

Run a Validator (Kusama)

This guide will instruct you how to set up a validator node on the Kusama network.

Preliminaries

Running a validator on a live network is a lot of responsibility! You will be accountable for not only your own stake, but also the stake of your current nominators. If you make a mistake and get slashed, your money and your reputation will be at risk. However, running a validator can also be very rewarding, knowing that you contribute to the security of a decentralized network while growing your stash.

Warning: It is highly recommended that you have significant system administration experience before attempting to run your own validator.

Since security is so important to running a successful validator, you should take a look at the secure validator information to make you understand the factors to consider when constructing your infrastructure. The Web3 Foundation also maintains a reference implementation for a secure validator set-up that you can use by deploying yourself. As you progress in your journey as a validator, you will likely want to use this repository as a *starting point* for your own modifications and customizations.

If you need help, please reach out on the Kusama validator chat on Matrix. The team and other validators are there to help answer questions and provide experience. You can join directly in your web browser (link above) or using a client such as Element (formerly Riot.im) using this link.

How Many KSM Do I Need?

You can have a rough estimate on that by using the methods listed here. Validators are elected based on Phragmén's algorithm. To be elected into the set, you need a minimum stake behind your validator. This stake can come from yourself or from nominators. This means that as a minimum, you will need enough KSM to set up Stash and Controller accounts with the existential deposit, plus a little extra for transaction fees. The rest can come from nominators.

Warning: Any KSM that you stake for your validator is liable to be slashed, meaning that an insecure or improper setup may result in loss of KSM tokens! If you are not confident in your ability to run a validator node, it is recommended to nominate your KSM to a trusted validator node instead.

Initial Set-up

Requirements

You will likely run your validator on a cloud server running Linux. You may choose whatever VPS provider that your prefer, and whatever operating system you are comfortable with. For this guide we will be using **Ubuntu 18.04**, but the instructions should be similar for other platforms.

You will not need a very powerful machine to run your validator, but you should be aware of the resource constraints. The most important resource for your validator node is networking bandwidth, followed by its storage and memory capabilities. The bare minimum requirements for a machine to run a validator are as follows:

- **Storage:** 160GB - 200GB. Kusama doesn't have very heavy storage requirements yet so something in this range will be fine, just keep in mind you may have to upgrade it later if the chain state becomes very big.
- **Memory:** 2GB - 8GB. 2GB is really the minimum memory you should operate your validator with, anything less than this make build times too inconvenient. For better performance you can bump it up to 4GB or 8GB, but anything more than that is probably over-kill. In order to compile the binary yourself you will likely need ~8GB.
- **CPU:** 1 - 2. One CPU is okay, but 2 is better. Again, this is a performance preference.

On most cloud service providers, these specs are usually within the $10 - $20 per month range.

Install Rust

Once you choose your cloud service provider and set-up your new server, the first thing you will do is install Rust.

If you have never installed Rust, you should do this first. This command will fetch the latest version of Rust and install it.

```
curl https://sh.rustup.rs -sSf | sh
```

Otherwise, if you have already installed Rust, run the following command to make sure you are using the latest version.

```
rustup update
```

Finally, run this command to install the necessary dependencies for compiling and running the Kusama node software.

```
sudo apt install make clang pkg-config libssl-dev build-essential
```

Note - if you are using OSX and you have Homebrew installed, you can issue the following equivalent command INSTEAD of the previous one:

```
brew install cmake pkg-config openssl git llvm
```

Install & Configure Network Time Protocol (NTP) Client

NTP is a networking protocol designed to synchronize the clocks of computers over a network. NTP allows you to synchronize the clocks of all the systems within the network. Currently it is required that validators' local clocks stay reasonably in sync, so you should be running NTP or a similar service. You can check whether you have the NTP client by running:

If you are using Ubuntu 18.04 / 19.04, NTP Client should be installed by default.

```
timedatectl
```

If NTP is installed and running, you should see `System clock synchronized: yes` (or a similar message). If you do not see it, you can install it by executing:

```
sudo apt-get install ntp
```

ntpd will be started automatically after install. You can query ntpd for status information to verify that everything is working:

```
sudo ntpq -p
```

Building and Installing the `polkadot` Binary

You will need to build the `polkadot` binary from the paritytech/polkadot repository on GitHub using the source code available in the **v0.8** branch.

You should generally use the latest **0.8.x** tag. At the time of writing, this was **0.8.26-1**, but you should review the output from the "git tag" command to see a list of all the potential 0.8 releases.

You should replace `VERSION` with the latest build (i.e., the highest number). You can also find the latest Kusama version on the release tab.

> Note: If you prefer to use SSH rather than HTTPS, you can replace the first line of the below with `git clone git@github.com:paritytech/polkadot.git`.

```
git clone https://github.com/paritytech/polkadot.git
cd polkadot
git tag -l | sort -V | grep -v -- '-rc'
echo Get the latest version and replace VERSION (below) with it.
git checkout VERSION
./scripts/init.sh
cargo build --release
```

This step will take a while (generally 10 - 40 minutes, depending on your hardware).

> Note if you run into compile errors, you may have to switch to a less recent nightly. This can be done by running:
>
> ```
> rustup install nightly-2020-05-15
> rustup override set nightly-2020-05-15
> rustup target add wasm32-unknown-unknown --toolchain nightly-2020-05-15
> ```

If you are interested in generating keys locally, you can also install `subkey` from the same directory. You may then take the generated `subkey` executable and transfer it to an air-gapped machine for extra security.

```
cargo install --force --git https://github.com/paritytech/substrate subkey
```

Synchronize Chain Data

> **Note:** By default, Validator nodes are in archive mode. If you've already synced the chain not in archive mode, you must first remove the database with `polkadot purge-chain` and then ensure that you run Polkadot with the `--pruning=archive` option.
>
> You may run a validator node in non-archive mode by adding the following flags: `--unsafe-pruning --pruning <NUMBER OF BLOCKS>`, but note that an archive node and non-archive node's databases are not compatible with each other, and to switch you will need to purge the chain data.

You can begin syncing your node by running the following command:

```
./target/release/polkadot --pruning=archive --chain kusama
```

if you do not want to start in validator mode right away.

The `--pruning=archive` flag is implied by the `--validator` and `--sentry` flags, so it is only required explicitly if you start your node without one of these two options. If you do not set your pruning to archive node, even when not running in validator and sentry mode, you will need to re-sync your database when you switch.

> **Note:** Validators should sync using the RocksDb backend. This is implicit by default, but can be explicit by passing the `--database RocksDb` flag. In the future, it is recommended to switch to using the faster and more efficient ParityDb option. Switching between database backends will require a resync.
>
> If you want to test out ParityDB you can add the flag `--database paritydb`.

Depending on the size of the chain when you do this, this step may take anywhere from a few minutes to a few hours.

If you are interested in determining how much longer you have to go, your server logs (printed to STDOUT from the `polkadot` process) will tell you the latest block your node has processed and verified. You can then compare that to the current highest block via Telemetry or the Polkadot-JS Block Explorer.

Bond KSM

It is highly recommended that you make your controller and stash accounts be two separate accounts. For this, you will create two accounts and make sure each of them have at least enough funds to pay the fees for making transactions. Keep most of your funds in the stash account since it is meant to be the custodian of your staking funds.

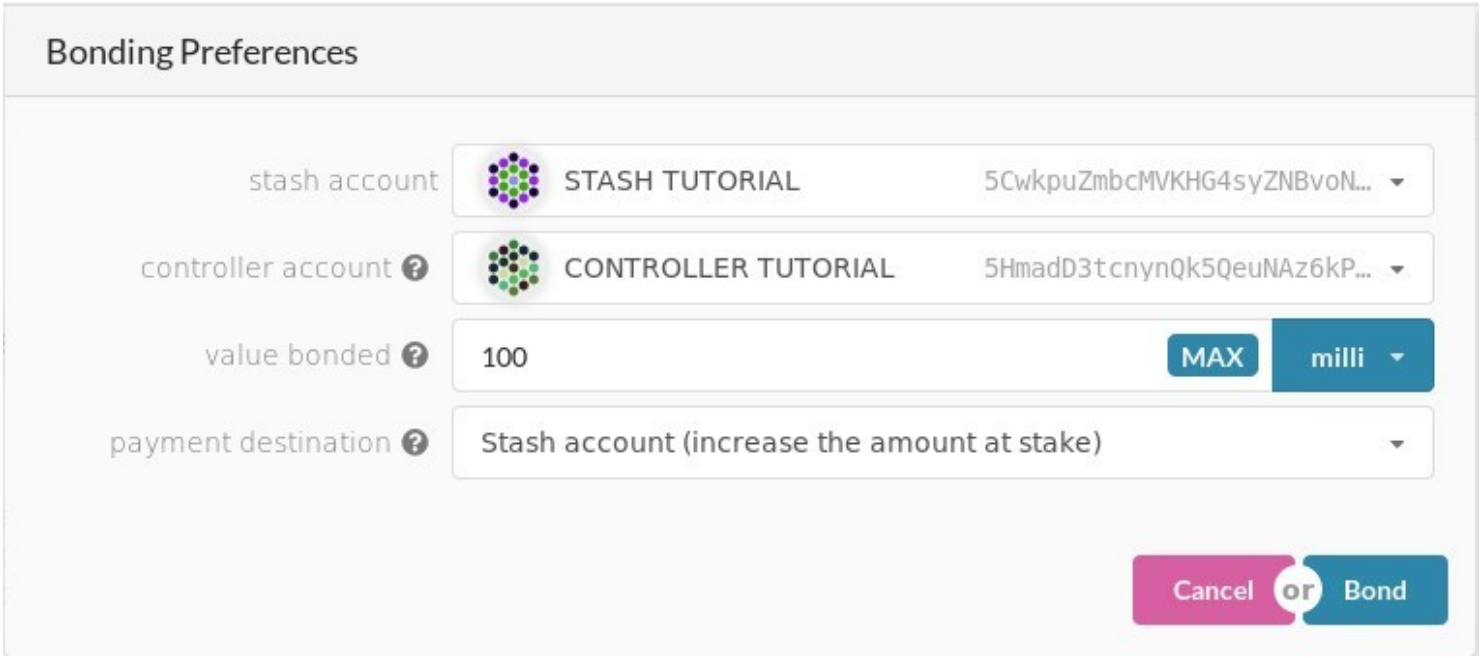
Make sure not to bond all your KSM balance since you will be unable to pay transaction fees from your bonded balance.

It is now time to set up our validator. We will do the following:

- Bond the KSM of the Stash account. These KSM will be put at stake for the security of the network and can be slashed.

- Select the Controller. This is the account that will decide when to start or stop validating.

First, go to the Staking section. Click on "Account Actions", and then the "New stake" button.



- **Stash account** - Select your Stash account. In this example, we will bond 100 milliKSM - make sure that your Stash account contains *at least* this much. You can, of course, stake more than this.
- **Controller account** - Select the Controller account created earlier. This account will also need a small amount of KSM in order to start and stop validating.
- **Value bonded** - How much KSM from the Stash account you want to bond/stake. Note that you do not need to bond all of the KSM in that account. Also note that you can always bond *more* KSM later. However, *withdrawing* any bonded amount requires the duration of the unbonding period. On Kusama, the unbonding period is 7 days. On Polkadot, the planned unbonding period is 28 days.
- **Payment destination** - The account where the rewards from validating are sent. More info here. Starting with runtime version v2023 natively included in client version 0.8.23, payouts can go to any custom address. If you'd like to redirect payments to an account that is neither the controller nor the stash account, set one up. Note that it is extremely unsafe to set an exchange address as the recipient of the staking rewards.

Once everything is filled in properly, click `Bond` and sign the transaction with your Stash account.

After a few seconds, you should see an "ExtrinsicSuccess" message. You should now see a new card with all your accounts (note: you may need to refresh the screen). The bonded amount on the right corresponds to the funds bonded by the Stash account.

Set Session Keys

> **Note:** The session keys are consensus critical, so if you are not sure if your node has the current session keys that you made the `setKeys` transaction then you can use one of the two available RPC methods to query your node: hasKey to check for a specific key or hasSessionKeys to check the full session key public key string.

Once your node is fully synced, stop the process by pressing Ctrl-C. At your terminal prompt, you will now start running the node in validator mode with a flag allowing unsafe RPC calls, needed for some advanced operations.

```
./target/release/polkadot --validator --name "name on telemetry" --chain kusama
```

You can give your validator any name that you like, but note that others will be able to see it, and it will be included in the list of all servers using the same telemetry server. Since numerous people are using telemetry, it is recommended that you choose something likely to be unique.

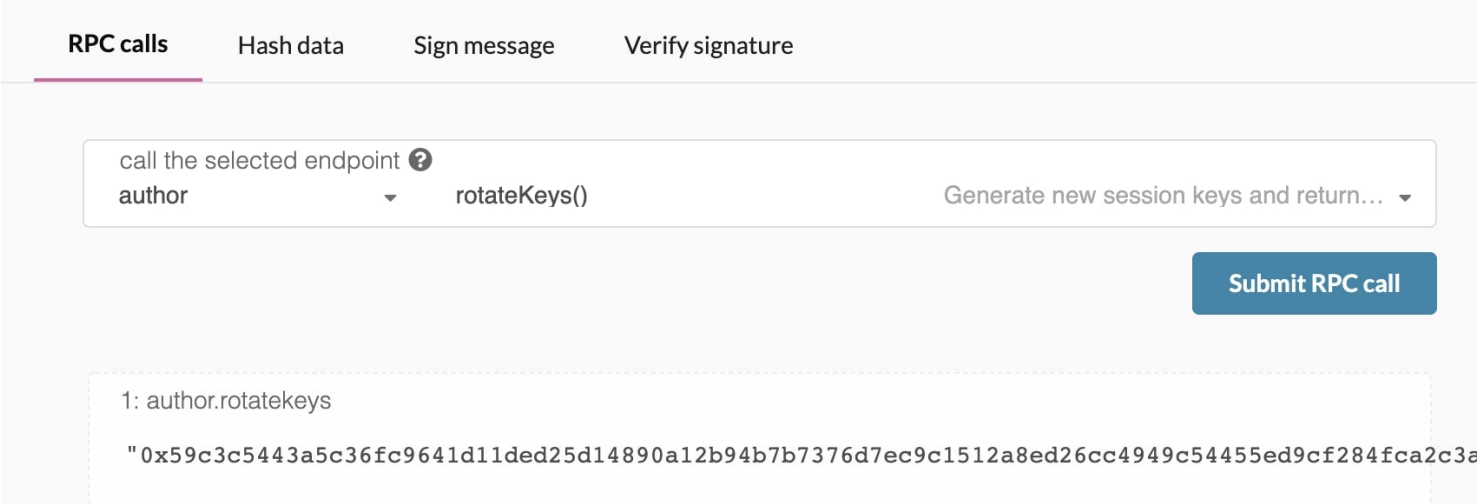
Generating the Session Keys

You need to tell the chain your Session keys by signing and submitting an extrinsic. This is what associates your validator node with your Controller account on Polkadot.

Option 1: Polkadot-JS Apps

You can generate your Session keys in the client via the apps RPC. If you are doing this, make sure that you have the Polkadot-JS Apps explorer attached to your validator node. You can configure the apps dashboard to connect to the endpoint of your validator in the Settings tab. If you are connected to a default endpoint hosted by Parity of Web3 Foundation, you will not be able to use this method since making RPC requests to this node would effect the local keystore hosted on a *public node* and you want to make sure you are interacting with the keystore for *your node*.

Once ensuring that you have connected to your node, the easiest way to set session keys for your node is by calling the `author_rotateKeys` RPC request to create new keys in your validator's keystore. Navigate to Toolbox tab and select RPC Calls then select the author > rotateKeys() option and remember to save the output that you get back for a later step.



Option 2: CLI

If you are on a remote server, it is easier to run this command on the same machine (while the node is running with the default HTTP RPC port configured):

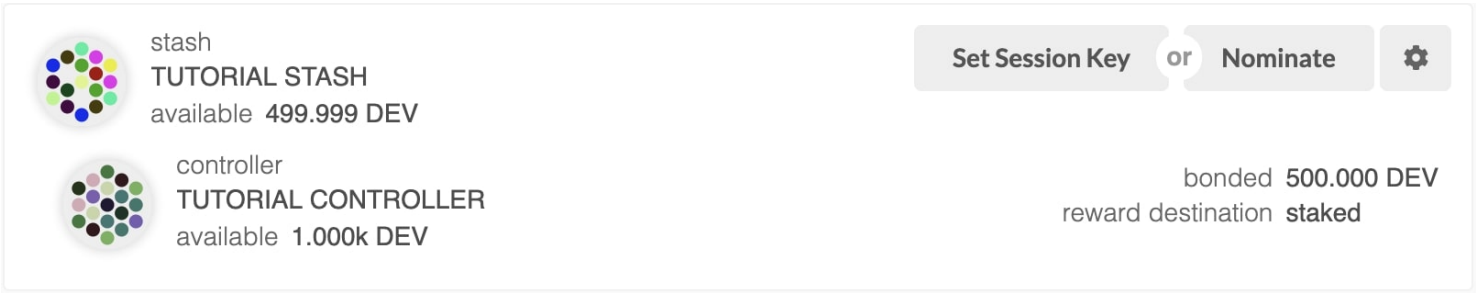
```
curl -H "Content-Type: application/json" -d '{"id":1, "jsonrpc":"2.0", "method": "author_rotateKeys", "params":[]}' http://localhost:9933
```

The output will have a hex-encoded "result" field. The result is the concatenation of the four public keys. Save this result for a later step.

Submitting the `setKeys` Transaction

You need to tell the chain your Session keys by signing and submitting an extrinsic. This is what associates your validator with your Controller account.

Go to Staking > Account Actions, and click "Set Session Key" on the bonding account you generated earlier. Enter the output from `author_rotateKeys` in the field and click "Set Session Key".

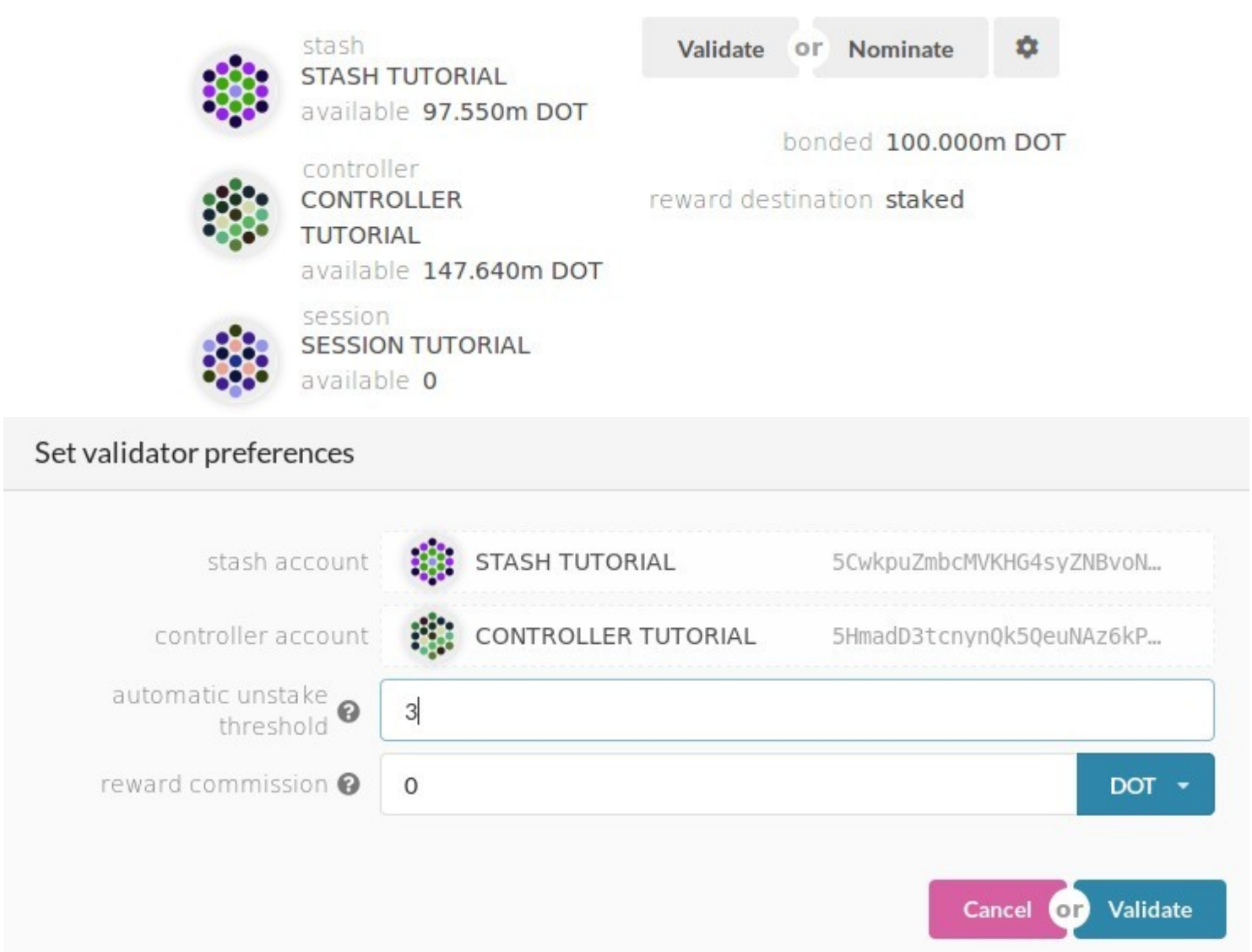


Submit this extrinsic and you are now ready to start validating.

Validate

To verify that your node is live and synchronized, head to Telemetry and find your node. Note that this will show all nodes on the Kusama network, which is why it is important to select a unique name!

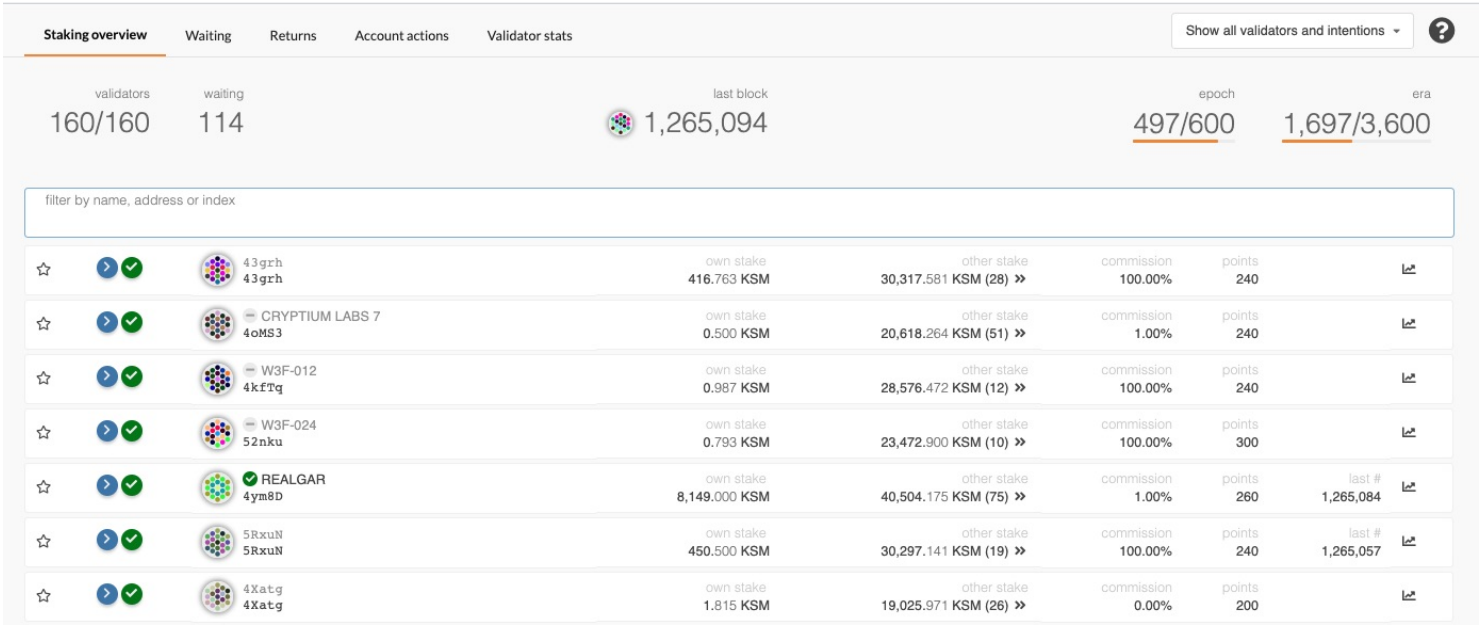
If everything looks good, go ahead and click on "Validate" in Polkadot-JS UI.



- **Payment preferences** - Rewards you will keep, the rest will be shared among you and your nominators.

Click "Validate".

If you go to the "Staking" tab, you will see a list of active validators currently running on the network. At the top of the page, it shows how many validator slots are available as well as how many nodes have signaled their intention to be a validator. You can also go to the "Waiting" tab to double check to see whether your node is listed there.



The validator set is refreshed every era. In the next era, if there is a slot available and your node is selected to join the validator set, your node will become an active validator. Until then, it will remain in the *waiting* queue. If your validator is not selected to become part of the validator set, it will remain in the *waiting* queue until it is. There is no need to re-start if you are not selected for the validator set in a particular era. However, it may be necessary to increase the number of KSM staked or seek out nominators for your validator in order to join the validator set.

Congratulations! If you have followed all of these steps, and been selected to be a part of the validator set, you are now running a Kusama validator! If you need help, reach out on the Kusama forum or in the Kusama Validator chat.

Thousand Validators Programme

The Thousand Validators Programme is a joint initiative by Web3 Foundation and Parity Technologies to provide support for community validators. If you are interested in applying for the programme, you can find more information on the wiki page.

FAQ

Why am I unable to synchronize the chain with 0 peers?

```
2019-08-23 19:40:45 Idle (0 peers), best: #0 (0x3fd7…5baf), finalized #0 (0x3fd7…5baf), ⬇ 2.9kiB/s ⬆ 3.7kiB/s
2019-08-23 19:40:50 Idle (0 peers), best: #0 (0x3fd7…5baf), finalized #0 (0x3fd7…5baf), ⬇ 1.7kiB/s ⬆ 2.0kiB/s
2019-08-23 19:40:55 Idle (0 peers), best: #0 (0x3fd7…5baf), finalized #0 (0x3fd7…5baf), ⬇ 0.9kiB/s ⬆ 1.2kiB/s
2019-08-23 19:40:57 Libp2p => Random Kademlia query has yielded empty results
2019-08-23 19:41:00 Idle (0 peers), best: #0 (0x3fd7…5baf), finalized #0 (0x3fd7…5baf), ⬇ 1.6kiB/s ⬆ 1.9kiB/s
2019-08-23 19:41:05 Idle (0 peers), best: #0 (0x3fd7…5baf), finalized #0 (0x3fd7…5baf), ⬇ 0.6kiB/s ⬆ 0.9kiB/s
2019-08-23 19:41:10 Idle (0 peers), best: #0 (0x3fd7…5baf), finalized #0 (0x3fd7…5baf), ⬇ 0.9kiB/s ⬆ 0.8kiB/s
2019-08-23 19:41:15 Idle (0 peers), best: #0 (0x3fd7…5baf), finalized #0 (0x3fd7…5baf), ⬇ 1.7kiB/s ⬆ 2.1kiB/s
```

Make sure to enable `30333` libp2p port. Eventually, it will take a little bit of time to discover other peers over the network.

How do I clear all my chain data?

```
./target/release/polkadot purge-chain
```

VPS List

- OVH
- Digital Ocean
- Vultr
- Linode
- Contabo
- Scaleway

Using Docker

If you have Docker installed, you can use it to start your validator node without needing to build the binary. You can do this with a simple one line command:

```
$ docker run parity/polkadot:latest --validator --name "name on telemetry" --
chain kusama
```

How to Stop Validating

If you wish to remain a validator or nominator (e.g. you're only stopping for planned downtime or server maintenance), submitting the `chill` extrinsic in the `staking` pallet should suffice. It is only if you wish to unbond funds or reap an account that you should continue with the following.

To ensure a smooth stop to validation, make sure you should do the following actions:

- Chill your validator
- Purge validator session keys
- Unbond your tokens

These can all be done with PolkadotJS Apps interface or with extrinsics.

Chill Validator

To chill your validator or nominator, call the `staking.chill()` extrinsic. See the How to Chill page for more information. You can also claim your rewards at this time.

Purge validator session keys

Purging the validator's session keys removes the key reference to your stash. This can be done through the `session.purgeKeys()` extrinsic with the controller account.

> NOTE: **If you skip this step, you will not be able to reap your stash account**, and you will need to rebond, purge the session keys, unbond, and wait the unbonding period again before being able to transfer your tokens. See Unbonding and Rebonding for more details.

Unbond your tokens

Unbonding your tokens can be done through the `Network > Staking > Account actions` page in PolkadotJS Apps by clicking the corrosponding stash account dropdown and selecting "Unbond funds". This can also be done through the `staking.unbond()` extrinsic with the controller account.

Be a Nominator (Kusama)

> The following information applies to the Kusama network. If you want to nominate on Polkadot, check out the Polkadot guide instead.

Nominators are one type of participant in the staking subsystem of Polkadot. They are responsible for appointing their stake to the validators who are the second type of participant. By appointing their stake, they are able to elect the active set of validators and share in the rewards that are paid out.

While the validators are active participants in the network that engage in the block production and finality mechanisms, nominators take a more passive role with a "set-it-and-forget-it" approach. Being a nominator does not require running a node of your own or worrying about online uptime. However, a good nominator performs due diligence on the validators that they elect. When looking for validators to nominate, a nominator should pay attention to their own reward percentage for nominating a specific validator - as well as the risk that they bear of being slashed if the validator gets slashed.

Setting up Stash and Controller keys

> If you prefer a video format, the following videos related to staking are available:
>
> - Staking with a Ledger and PolkadotJS Apps
> - Staking with a Ledger and Ledger Live

Nominators are recommended to set up two separate stash and controller accounts. Explanation and reasoning for generating distinct accounts for this purpose is elaborated in the keys section of the Wiki.

You can generate your stash and controller account via any of the recommended methods that are detailed on the account generation page.

Starting with runtime version v2023 natively included in client version 0.8.23, payouts can go to any custom address. If you'd like to redirect payments to an account that is neither the controller nor the stash account, set one up. Note that it is extremely unsafe to set an exchange address as the recipient of the staking rewards.

Using Polkadot-JS UI

Step 1: Bond your tokens

On the Polkadot-JS UI navigate to the "Staking" tab (within the "Network" menu).

The "Staking Overview" subsection will show you all the active validators and their information - their identities, the amount of KSM that are staking for them, amount that is their own provided stake, how much they charge in commission, the era points they've earned in the current era, and the last block number that they produced. If you click on the chart button it will take you to the "Validator Stats" page for that validator that shows you more detailed and historical information about the validator's stake, rewards and slashes.

The "Account actions" subsection (link) allows you to stake and nominate.

The "Payouts" subsection (link) allows you to claim rewards from staking.

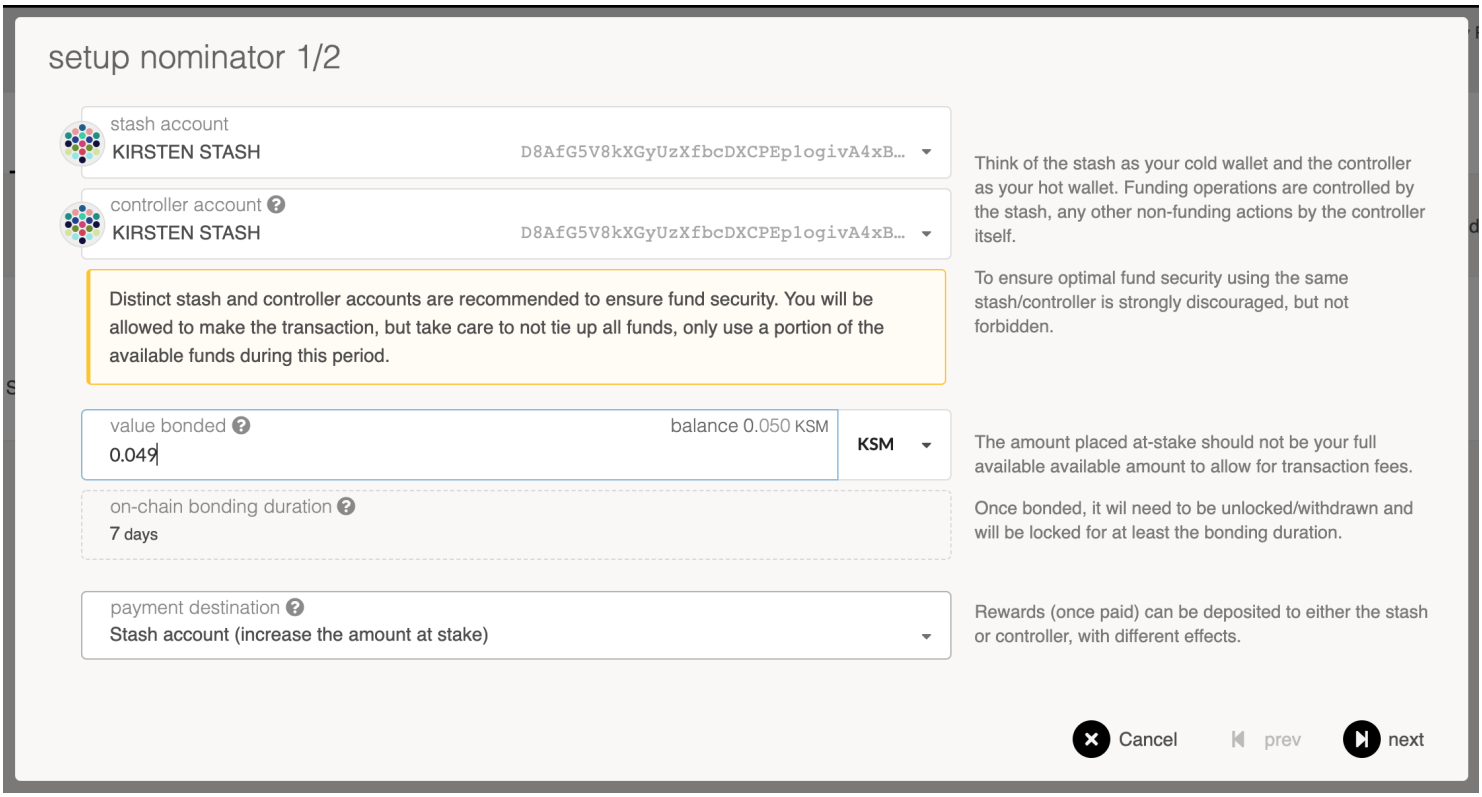
The "Targets" subsection (link) will help you estimate your earnings and this is where it's good to start picking favorites.

The "Waiting" subsection (link) lists all pending validators that are awaiting more nominations to enter the active validator set. Validators will stay in the waiting queue until they have enough KSM backing them (as allocated through the Phragmén election mechanism). It is possible validator can remain in the queue for a very long time if they never get enough backing.

The "Validator Stats" subsection (link) allows you to query a validator's stash address and see historical charts on era points, elected stake, rewards, and slashes.

Pick "Account actions" underneath "Network" > "Staking", then click the "+ Nominator" button.

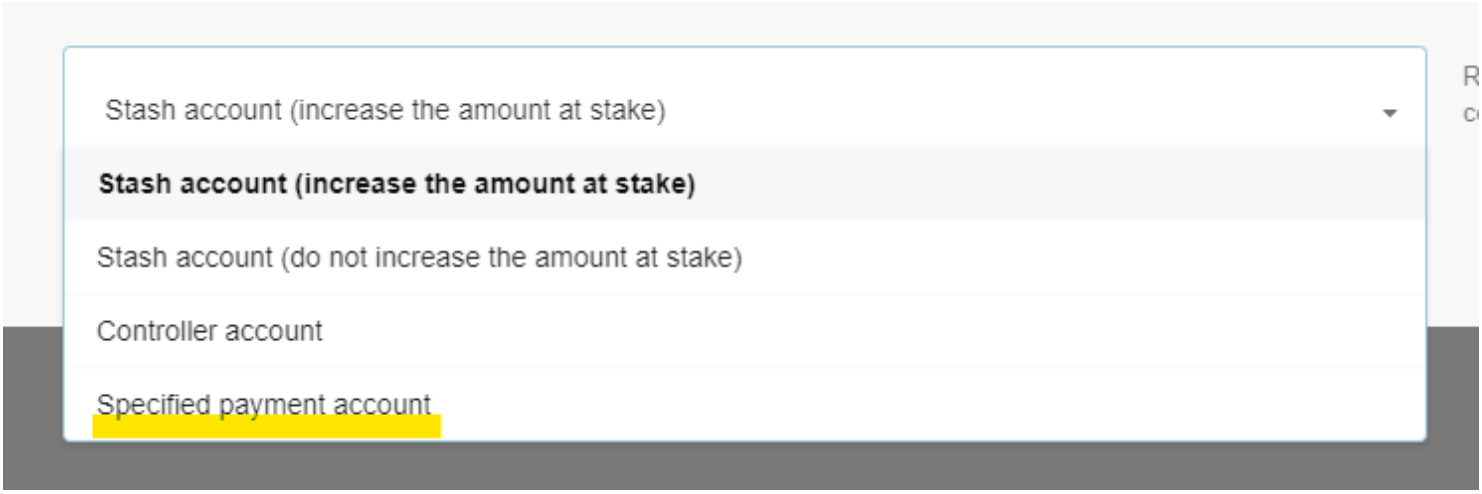
You will see a modal window that looks like the below:



Select a "value bonded" that is **less** than the total amount of KSM you have, so you have some left over to pay transaction fees. Transaction fees are currently at least 0.01 KSM, but they are dynamic based on a variety of factors including the load of recent blocks.

Also be mindful of the reaping threshold - the amount that must remain in an account lest it be burned. That amount is 0.01 in Kusama, so it's recommended to keep at least 0.1 KSM in your account to be on the safe side.

Choose whatever payment destination that makes sense to you. If you're unsure, you can choose "Stash account (increase amount at stake)" to simply accrue the rewards into the amount you're staking and earn compound interest.

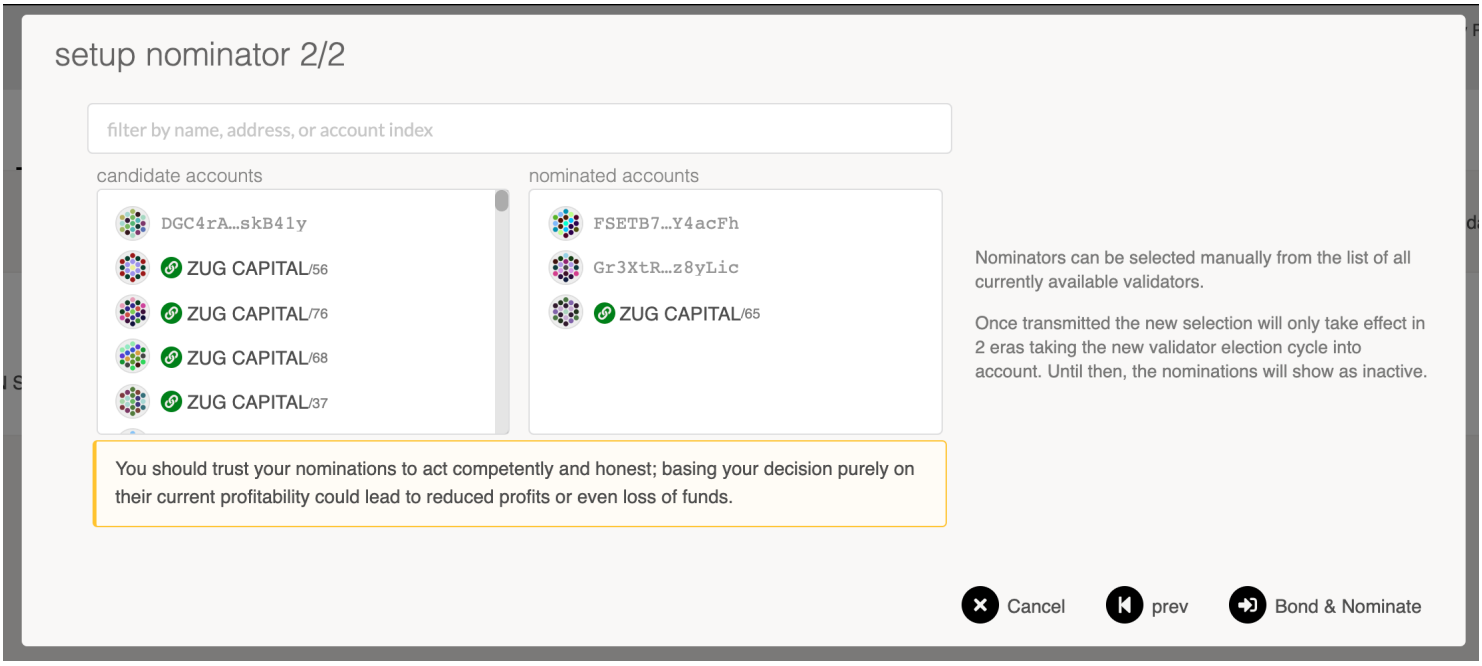


> These concepts have been further explained in Polkadot's UI Walkthrough Video

Step 2: Nominate a validator

You are now bonded. Being bonded means your tokens are locked and could be slashed if the validators you nominate misbehave. All bonded funds can now be distributed to up to 16 validators. Be careful about the validators you choose since you will be slashed if your validator commits an offence.

Click on "Nominate" on an account you've bonded and you will be presented with another popup asking you to select some validators.



Select them, confirm the transaction, and you're done - you are now nominating. Your nominations will become active in the next era. Eras last six hours on Kusama - depending on when you do this, your nominations may become active almost immediately, or you may have to wait almost the entire six hours before your nominations are active. You can check how far along Kusama is in the current era on the Staking page.

Assuming at least one of your nominations ends up in the active validator set, you will start to get rewards allocated to you. In order to claim them (i.e., add them to your account), you must manually claim them. To initiate a claim, you can do it yourself or have the validator that you staked for initiate a claim. This is to help optimize the effectiveness and storage of payouts on Kusama. See the Claiming Rewards section of the Staking wiki page for more details.

Step 3: Stop nominating

At some point, you might decide to stop nominating one or more validators. You can always change who you're nominating, but you cannot withdraw your tokens unless you unbond them. Detailed instructions are available here.

Using Command-Line Interface (CLI)

Apart from using the Polkadot-JS UI to participate in the staking, you can do all these things in CLI instead. The CLI approach allows you to interact with the Polkadot / Kusama network without going to the PolkadoJS dashboard.

Step 1: Install @polkadot/api-cli

We assume you have installed NodeJS with npm. Run the following command to install the `@polkadot/api-cli` globally:

```
npm install -g @polkadot/api-cli
```

Step 2. Bond your KSM

Executing the following command:

```
polkadot-js-api --seed "MNEMONIC_PHRASE" tx.staking.bond CONTROLLER_ADDRESS
NUMBER_OF_TOKENS REWARD_DESTINATION --ws WEBSOCKET_ENDPOINT
```

`CONTROLLER_ADDRESS`: An address you would like to bond to the stash account. Stash and Controller can be the same address but it is not recommended since it defeats the security of the two-account staking model.

`NUMBER_OF_TOKENS`: The number of KSM / DOT you would like to stake to the network. **Note**: KSM has twelve decimal places and is always represented as an integer with zeroes at the end. So 1 KSM = 1_000_000_000_000 Plancks.

`REWARD_DESTINATION`:

- `Staked` - Pay into the stash account, increasing the amount at stake accordingly.
- `Stash` - Pay into the stash account, not increasing the amount at stake.
- `Account` - Pay into a custom account, like so: `Account DMTHrNcmA8QbqRS4rBq8LXn8ipyczFoNMb1X4cY2WD9tdBX`.
- `Controller` - Pay into the controller account.

Example:

```
polkadot-js-api --seed "xxxx xxxxx xxxx xxxxx" tx.staking.bond
DMTHrNcmA8QbqRS4rBq8LXn8ipyczFoNMb1X4cY2WD9tdBX 1000000000000 Staked --ws
```

```
wss://kusama-rpc.polkadot.io/
```

Result:

```
...
...
    "status": {
      "InBlock":
"0x0ed1ec0ba69564e8f98958d69f826adef895b5617366a32a3aa384290e98514e"
    }
```

You can check the transaction status by using the value of the `InBlock` in Polkascan. Also, you can verify the bonding state under the Staking page on the Polkadot-JS Apps Dashboard.

Step 3. Nominate a validator

To nominate a validator, you can execute the following command:

```
polkadot-js-api --seed "MNEMONIC_PHRASE" tx.staking.nominate
'["VALIDATOR_ADDRESS"]' --ws WS_ENDPOINT
```

```
polkadot-js-api --seed "xxxx xxxxx xxxx xxxxx" tx.staking.nominate
'["CmD9vaMYoiKe7HiFnfkftwvhKbxN9bhyjcDrfFRGbifJEG8","E457XaKbj2yTB2URy8N4UuzmyuFRkc
--ws wss://kusama-rpc.polkadot.io/
```

After a few seconds, you should see the hash of the transaction and if you would like to verify the nomination status, you can check that on the Polkadot-JS UI as well.

Governance

Polkadot uses a sophisticated governance mechanism that allows it to evolve gracefully over time at the ultimate behest of its assembled stakeholders. The stated goal is to ensure that the majority of the stake can always command the network.

To do this, we bring together various novel mechanisms, including an amorphous state-transition function stored on-chain and defined in a platform-neutral intermediate language (i.e. WebAssembly) and several on-chain voting mechanisms such as referenda with adaptive super-majority thresholds and batch approval voting. All changes to the protocol must be agreed upon by stake-weighted referenda.

Mechanism

In order to make any changes to the network, the idea is to compose active token holders and the council together to administrate a network upgrade decision. No matter whether the proposal is proposed by the public (token holders) or the council, it finally will have to go through a referendum to let all holders, weighted by stake, make the decision.

To better understand how the council is formed, please read this section.

Referenda

Referenda are simple, inclusive, stake-based voting schemes. Each referendum has a specific *proposal* associated with it that takes the form of a privileged function call in the runtime (that includes the most powerful call: `set_code`, which is able to switch out the entire code of the runtime, achieving what would otherwise require a "hard fork").

Referenda are discrete events, have a fixed period where voting happens, and then are tallied and the function call is made if the vote is approved. Referenda are always binary; your only options in voting are "aye", "nay", or abstaining entirely.

Referenda can be started in one of several ways:

- Publicly submitted proposals;
- Proposals submitted by the council, either through a majority or unanimously;
- Proposals submitted as part of the enactment of a prior referendum;
- Emergency proposals submitted by the Technical Committee and approved by the Council.

All referenda have an *enactment delay* associated with them. This is the period of time between the referendum ending and, assuming the proposal was approved, the changes being enacted. For the first two ways that a referendum is launched, this is a fixed time. For Kusama, it is 8 days; in Polkadot, it is 28 days. For the third type, it can be set as desired.

Emergency proposals deal with major problems with the network that need to be "fast-tracked". These will have a shorter enactment time.

Proposing a Referendum

Public Referenda

Anyone can propose a referendum by depositing the minimum amount of tokens for a certain period (number of blocks). If someone agrees with the proposal, they may deposit the same amount of tokens to support it - this action is called *seconding*. The proposal with the highest amount of bonded support will be selected to be a referendum in the next voting cycle.

Note that this may be different than the absolute number of seconds; for instance, three accounts bonding 20 DOT each would "outweigh" ten accounts bonding a single DOT each. The bonded tokens will be released once the proposal is tabled (that is, brought to a vote).

There can be a maximum of 100 public proposals in the proposal queue.

Council Referenda

Unanimous Council - When all members of the council agree on a proposal, it can be moved to a referendum. This referendum will have a negative turnout bias (that is, the smaller the amount of stake voting, the smaller the amount necessary for it to pass - see "Adaptive Quorum Biasing", below).

Majority Council - When agreement from only a simple majority of council members occurs, the referendum can also be voted upon, but it will be majority-carries (51% wins).

There can only be one active referendum at any given time, except when there is also an emergency referendum in progress.

Voting Timetable

Every 28 days on Polkadot or 7 days on Kusama, a new referendum will come up for a vote, assuming there is at least one proposal in one of the queues. There is a queue for Council-approved proposals and a queue for publicly submitted proposals. The referendum to be voted upon alternates between the top proposal in the two queues.

The "top" proposal is determined by the amount of stake bonded behind it. If the given queue whose turn it is to create a referendum has no proposals (is empty), and there are proposals waiting in the other queue, the top proposal in the other queue will become a referendum.

Multiple referenda cannot be voted upon in the same time period, excluding emergency referenda. An emergency referendum occurring at the same time as a regular referendum (either public- or council-proposed) is the only time that multiple referenda will be able to be voted on at once.

Voting on a referendum

To vote, a voter generally must lock their tokens up for at least the enactment delay period beyond the end of the referendum. This is in order to ensure that some minimal economic buy-in to the result is needed and to dissuade vote selling.

It is possible to vote without locking at all, but your vote is worth a small fraction of a normal vote, given your stake. At the same time, holding only a small amount of tokens does not mean that the holder cannot influence the referendum result, thanks to time-locking. You can read more about this at Voluntary Locking.

> To learn more about voting on referenda, please check out our technical explainer video.

```
Example:

Peter: Votes `No` with 10 DOT for a 128 week lock period  => 10 * 6 = 60 Votes

Logan: Votes `Yes` with 20 DOT for a 4 week lock period => 20 * 1 = 20 Votes

Kevin: Votes `Yes` with 15 DOT for a 8 week lock period => 15 * 2 = 30 Votes
```

Even though combined both Logan and Kevin vote with more DOT than Peter, the lock period for both of them is less than Peter, leading to their voting power counting as less.

Tallying

Depending on which entity proposed the proposal and whether all council members voted yes, there are three different scenarios. We can use following table for reference.

Entity	Metric
Public	Positive Turnout Bias (Super-Majority Approve)
Council (Complete agreement)	Negative Turnout Bias (Super-Majority Against)

Entity	Metric
Council (Majority agreement)	Simple Majority

Also, we need the following information and apply one of the formulas listed below to calculate the voting result. For example, let's use the public proposal as an example, so the `Super-Majority Approve` formula will be applied. There is no strict quorum, but the super-majority required increases as turnout lowers.

```
approve - the number of aye votes

against - the number of nay votes

turnout - the total number of voting tokens (does not include conviction)

electorate - the total number of DOT tokens issued in the network
```

Super-Majority Approve

A `positive turnout bias`, whereby a heavy super-majority of aye votes is required to carry at low turnouts, but as turnout increases towards 100%, it becomes a simple majority-carries as below.

$$\frac{against}{\sqrt{turnout}} < \frac{approve}{\sqrt{electorate}}$$

Super-Majority Against

A `negative turnout bias`, whereby a heavy super-majority of nay votes is required to reject at low turnouts, but as turnout increases towards 100%, it becomes a simple majority-carries as below.

$$\frac{against}{\sqrt{electorate}} < \frac{approve}{\sqrt{turnout}}$$

Simple-Majority

Majority-carries, a simple comparison of votes; if there are more aye votes than nay, then the proposal is carried, no matter how much stake votes on the proposal.

$$approve > against$$

To know more about where these above formulas come from, please read the democracy pallet.

```
Example:

Assume:
- We only have 1_500 DOT tokens in total.
- Public proposal

John  - 500 DOT
Peter - 100 DOT
Lilly - 150 DOT
JJ    - 150 DOT
Ken   - 600 DOT

John: Votes `Yes` for a 4 week lock period  => 500 * 1 = 500 Votes

Peter: Votes `Yes` for a 4 week lock period => 100 * 1 = 100 Votes

JJ: Votes `No` for a 16 week lock period => 150 * 3 = 450 Votes

approve = 600
against = 450
turnout = 750
electorate = 1500
```

$$\frac{450}{\sqrt{750}} < \frac{600}{\sqrt{1500}}$$

$$16.432 < 15.492$$

Since the above example is a public referendum, `Super-Majority Approve` would be used to calculate the result. `Super-Majority Approve` requires more `aye` votes to pass the referendum when turnout is low, therefore, based on the above result, the referendum will be rejected. In addition, only the winning voter's tokens are locked. If the voters on the losing side of the referendum believe that the outcome will have negative effects, their tokens are transferrable so they will not be locked in to the decision. Moreover, winning proposals are autonomously enacted only after some enactment period.

Voluntary Locking

Polkadot utilizes an idea called `Voluntary Locking` that allows token holders to increase their voting power by declaring how long they are willing to lock-up their tokens, hence, the number of votes for each token holder will be calculated by the following formula:

```
votes = tokens * conviction_multiplier
```

The conviction multiplier increases the vote multiplier by one every time the number of lock periods double.

Lock Periods	Vote Multiplier
0	0.1
1	1
2	2
4	3
8	4
16	5
32	6

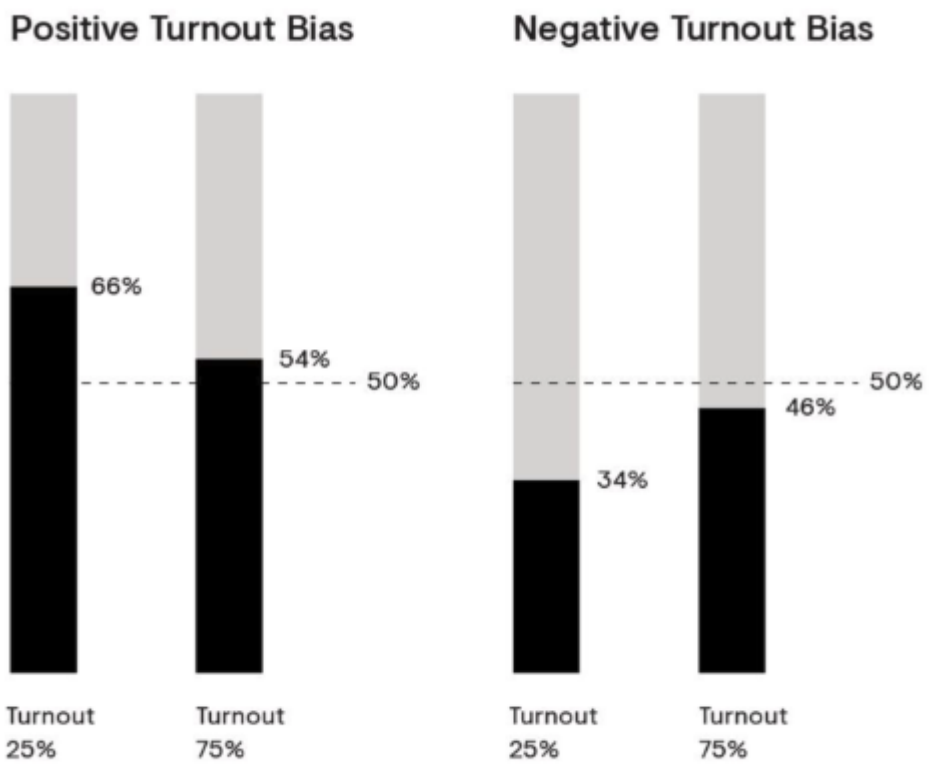
The maximum number of "doublings" of the lock period is set to 6 (and thus 32 lock periods in total), and one lock period equals 28 days on Polkadot and 8 days on Kusama. Only doublings are allowed; you cannot lock for, say, 24 periods and increase your conviction by 5.5, for instance.

While a token is locked, you can still use it for voting and staking; you are only prohibited from transferring these tokens to another account.

Votes are still "counted" at the same time (at the end of the voting period), no matter for how long the tokens are locked.

Adaptive Quorum Biasing

Polkadot introduces a concept, "Adaptive Quorum Biasing", which functions as a lever that the council can use to alter the effective super-majority required to make it easier or more difficult for a proposal to pass in the case that there is no clear majority of voting power backing it or against it.



Let's use the above image as an example.

If a publicly submitted referendum only has 25% turnout, the tally of "aye" votes has to reach 66% for it to pass since we applied `Positive Turnout Bias`.

In contrast, when it has 75% turnout, the tally of "aye" votes has to reach 54%, which means that the super-majority required decreases as the turnout increases.

When the council proposes a new proposal through unanimous consent, the referendum would be put to a vote using "Negative Turnout Bias". In this case it is easier to pass this proposal with low turnout and requires a super-majority to reject. As more token holders participate in voting, the bias approaches a plain majority carries.

Referring to the above image, when a referendum only has 25% turnout, the tally of "aye" votes has to reach 34% for it to pass.

In short, when turnout rate is low, a super-majority is required to reject the proposal, which means a lower threshold of "aye" votes have to be reached, but as turnout increases towards 100%, it becomes a simple-majority.

All three tallying mechanisms - majority carries, super-majority approve, and super-majority against - equate to a simple majority-carries system at 100% turnout.

Council

> Video explainer on Council

To represent passive stakeholders, Polkadot introduces the idea of a "council". The council is an on-chain entity comprising a number of actors, each represented as an on-chain account. On Polkadot, the council currently consists of 13 members. This is expected to increase over the next few months to 24 seats. In general, the council will end up having a fixed number of seats. On Polkadot, this will be 24 seats while on Kusama it is 19 seats.

Along with controlling the treasury, the council is called upon primarily for three tasks of governance: proposing sensible referenda, cancelling uncontroversially dangerous or malicious referenda, and electing the technical committee.

For a referendum to be proposed by the council, a strict majority of members must be in favor, with no member exercising a veto. Vetoes may be exercised only once by a member for any single proposal; if, after a cool-down period, the proposal is resubmitted, they may not veto it a second time.

Council motions which pass with a 3/5 (60%) super-majority - but without reaching unanimous support - will move to a public referendum under a neutral, majority-carries voting scheme. In the case that all members of the council vote in favor of a motion, the vote is considered unanimous and becomes a referendum with negative adaptive quorum biasing.

Canceling

A proposal can be canceled if the technical committee unanimously agrees to do so, or if Root origin (e.g. sudo) triggers this functionality. A canceled proposal's deposit is burned.

Additionally, a two-thirds majority of the council can cancel a referendum. This may function as a last-resort if there is an issue found late in a referendum's proposal such as a bug in the code of the runtime that the proposal would institute.

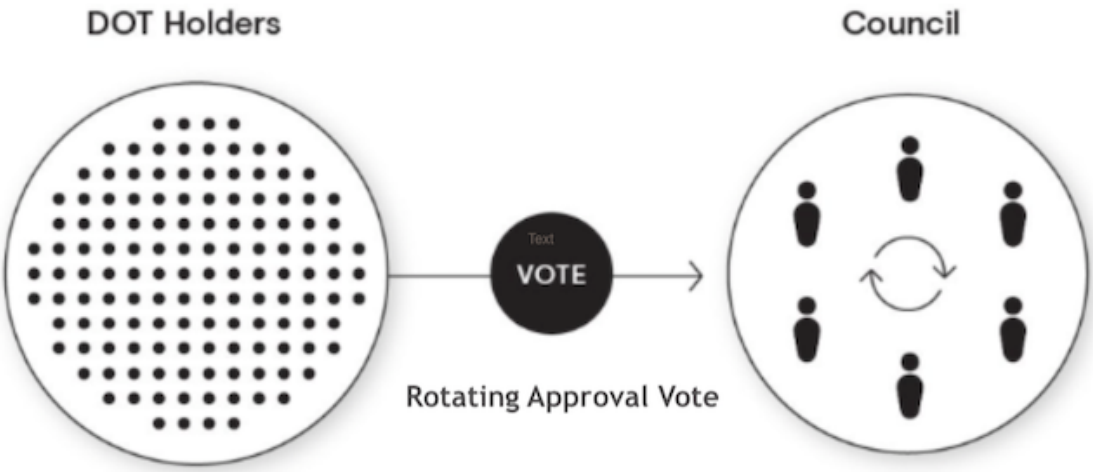
If the cancellation is controversial enough that the council cannot get a two-thirds majority, then it will be left to the stakeholders *en masse* to determine the fate of the proposal.

Blacklisting

A proposal can be blacklisted by Root origin (e.g. sudo). A blacklisted proposal and its related referendum (if any) is immediately canceled. Additionally, a blacklisted proposal's hash cannot re-appear in the proposal queue. Blacklisting is useful when removing erroneous proposals that could be submitted with the same hash, i.e. proposal #2 in which the submitter used plain text to make a suggestion.

Upon seeing their proposal removed, a submitter who is not properly introduced to the democracy system of Polkadot might be tempted to re-submit the same proposal. That said, this is far from a fool-proof method of preventing invalid proposals from being submitted - a single changed character in a proposal's text will also change the hash of the proposal, rendering the per-hash blacklist invalid.

How to be a council member?



All stakeholders are free to signal their approval of any of the registered candidates.

Council elections are handled by the same Phragmén election process that selects validators from the available pool based on nominations. However, token holders' votes for councillors are isolated from any of the nominations they may have on validators. Council terms last for one day on Kusama and one week on Polkadot.

At the end of each term, Phragmén election algorithm runs and the result will choose the new councillors based on the vote configurations of all voters. The election also chooses a set number of runners up (currently 19 on Kusama and 20 on Polkadot) that will remain in the queue with their votes intact.

As opposed to a "first-past-the-post" electoral system, where voters can only vote for a single candidate from a list, a Phragmén election is a more expressive way to include each voters' views. Token holders can treat it as a way to support as many candidates as they want. The election algorithm will find a fair subset of the candidates that most closely matches the expressed indications of the electorate as a whole.

Let's take a look at the example below.

Round 1					

Round 1					
Token Holders		**Candidates**			
	A	B	C	D	E
Peter	X		X	X	X
Alice		X			
Bob			X	X	X
Kelvin	X		X		
Total	2	1	3	2	2

The above example shows that candidate C wins the election in round 1, while candidate A, B, D & E keep remaining on the candidates' list for the next round.

Round 2				
Token Holders		**Candidates**		
	A	B	D	E
Peter	X	X		
Alice	X	X		
Bob	X	X	X	X
Kelvin	X	X		
Total	4	4	1	1

For the top-N (say 4 in this example) runners-up, they can remain and their votes persist until the next election. After round 2, even though candidates A & B get the same number of votes in this round, candidate A gets elected because after adding the older unused approvals, it is higher than B.

Prime Members

The council, being an instantiation of Substrate's Collective pallet, implements what's called a *prime member* whose vote acts as the default for other members that fail to vote before the timeout.

The prime member is chosen based on a Borda count.

The purpose of having a prime member of the council is to ensure a quorum, even when several members abstain from a vote. Council members might be tempted to vote a "soft rejection" or a "soft approval" by not voting and letting the others vote. With the existence of a prime member, it forces councillors to be explicit in their votes or have their vote counted for whatever is voted on by the prime.

Technical Committee

The Technical Committee was introduced in the Kusama rollout and governance post as one of the three chambers of Kusama governance (along with the Council and the Referendum chamber). The Technical Committee is composed of the teams that have successfully implemented or specified either a Polkadot/Kusama runtime or Polkadot Host. Teams are added or removed from the Technical Committee via simple majority vote of the Council.

The Technical Committee can, along with the Council, produce emergency referenda, which are fast-tracked for voting and implementation. These are used for emergency bug fixes, or rapid implementation of new but battle-tested features into the runtime.

Fast-tracked referenda are the only type of referenda that can be active alongside another active referendum. Thus, with fast tracked referenda it is possible to have two active referendums at the same time. Voting on one does not prevent a user from voting on the other.

Frequently Asked Questions

How can I appeal to the council to enact a change on my behalf?

In some circumstances you may want to appeal to the on-chain council to enact a change on your behalf. One example of this circumstance is the case of lost or locked funds when the funds were lost due to a human interface error (such as inputting an address for another network). Another example is if you participated in the 2017 Polkadot ICO with a multi-sig address which now does not let you sign a message easily. When these circumstances can be proven beyond a reasonable doubt to be an error, the council *may* consider a governance motion to correct it.

The first step to appeal to the council is to get in contact with the councillors. There is no singular place where you are guaranteed to grab every councillor's ear with your message. However, there are a handful of good places to start where you can get the attention of some of them. The Polkadot Direction matrix room is one such place. After creating an account and joining this room, you can post a well thought-through message here that lays down your case and provides justification for why you think the council should consider enacting a change to the protocol on your behalf.

At some point you will likely need a place for a longer form discussion. For this, making a post on Polkassembly is the recommended place to do so. When you write a post on Polkassembly make sure you present all the evidence for your circumstances and state clearly what kind of change you would suggest to the councillors to enact. Remember - the councillors do not need to make the change, it is your responsibility to make a strong case for why the change should be made.

Resources

- Initial Governance Description
- Democracy Pallet
- Governance Demo - Dr. Gavin Wood presents the initial governance structure for Polkadot. (Video)
- Governance on Polkadot - A webinar explaining how governance works in Polkadot and Kusama.

Identity

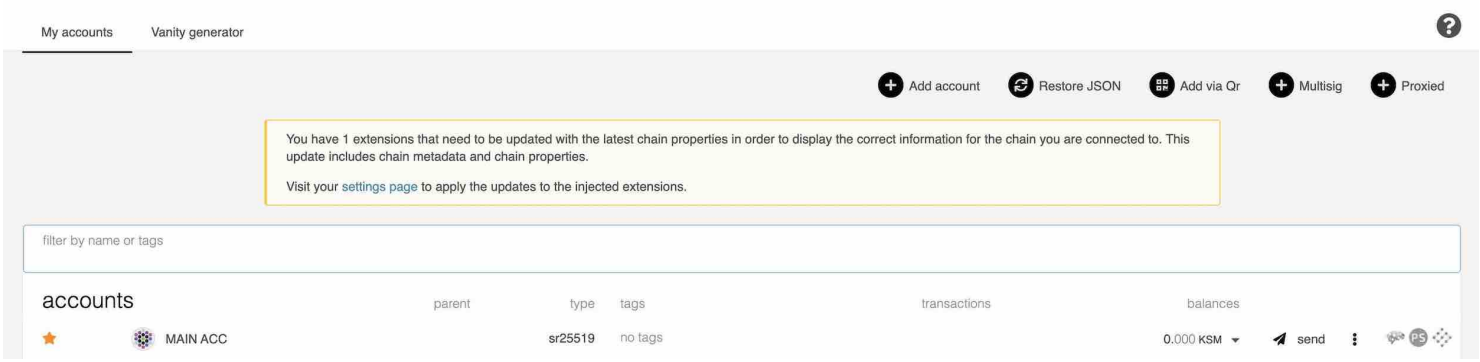
Polkadot provides a naming system that allows participants to add personal information to their on-chain account and subsequently ask for verification of this information by registrars.

Setting an Identity

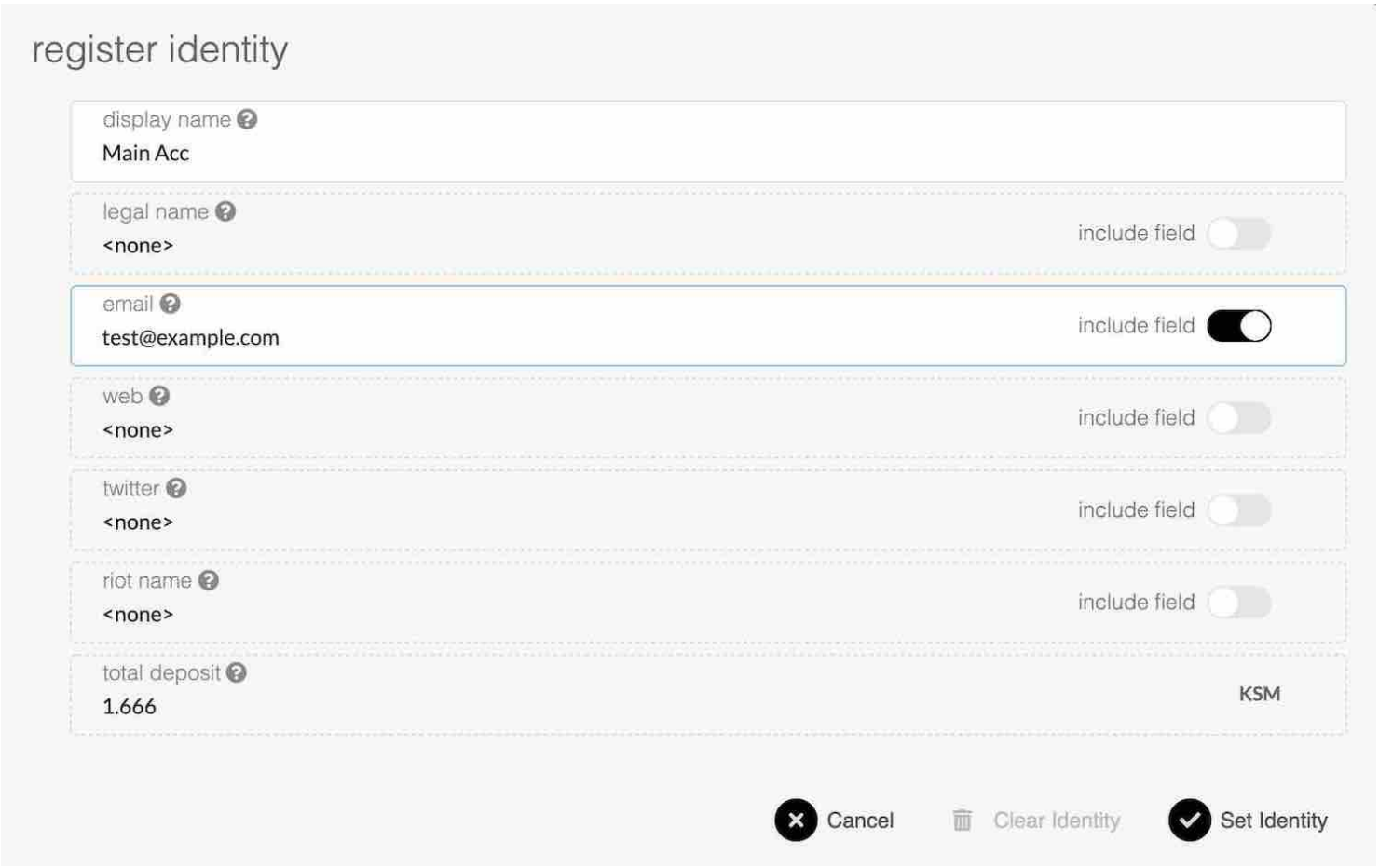
Users can set an identity by registering through default fields such as legal name, display name, website, Twitter handle, Riot handle, etc. along with some extra, custom fields for which they would like attestations (see Judgements).

Users must reserve funds in a bond to store their information on chain: {{ identity*reserve_funds }}, and {{ identity_field_funds }} per each field beyond the legal name. These funds are _locked*, not spent - they are returned when the identity is cleared. Each field can store up to 32 bytes of information, so the data must be less than that. When inputting the data manually through the Extrinsics UI, a UTF8 to bytes converter can help.

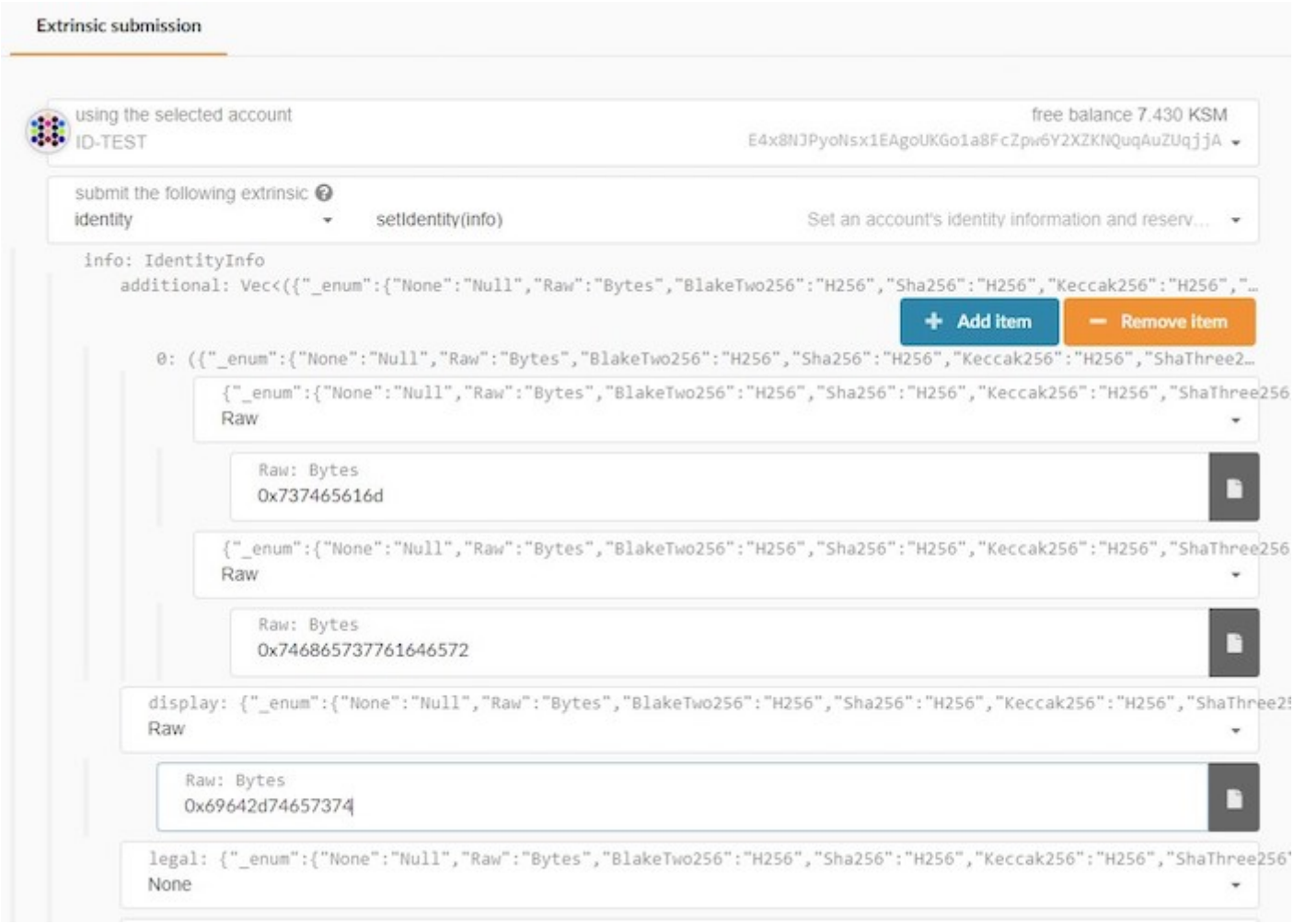
The easiest way to add the built-in fields is to click the gear icon next to your account and select "Set on-chain identity".



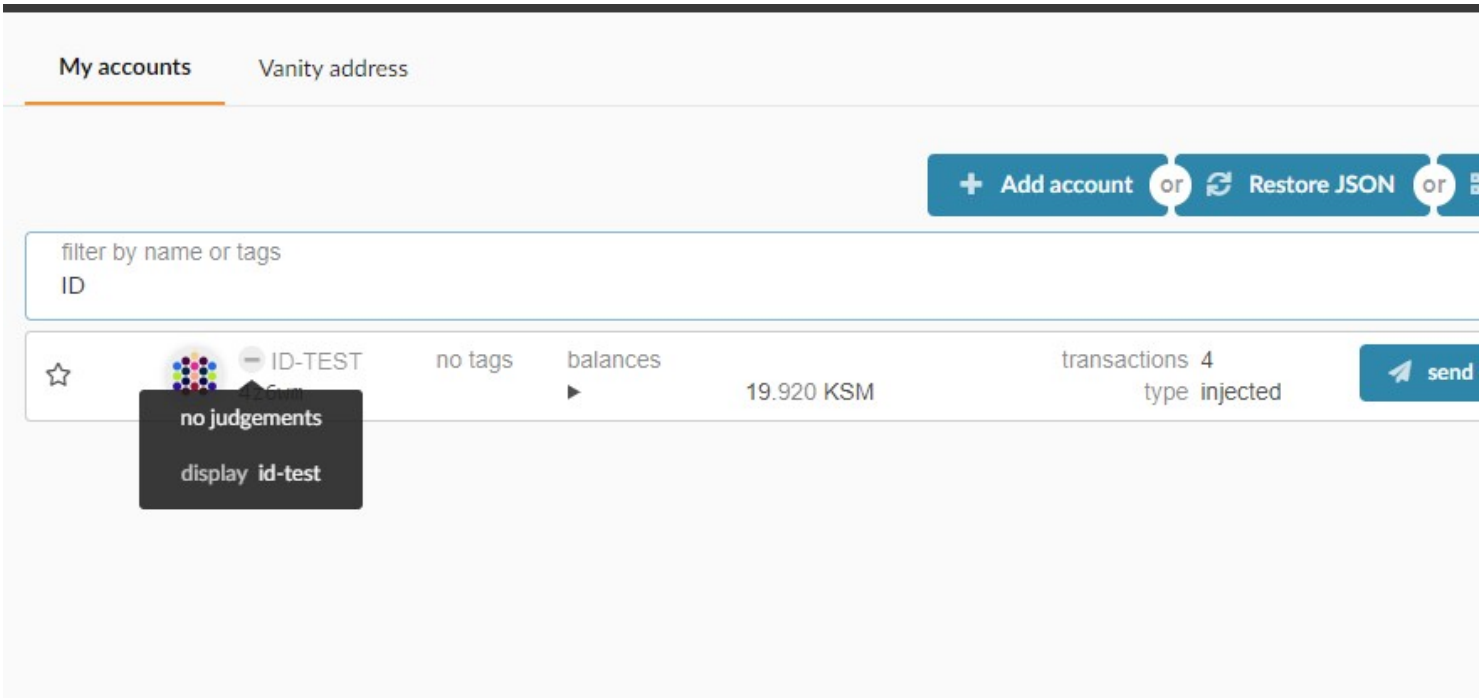
A popup will appear, offering the default fields.



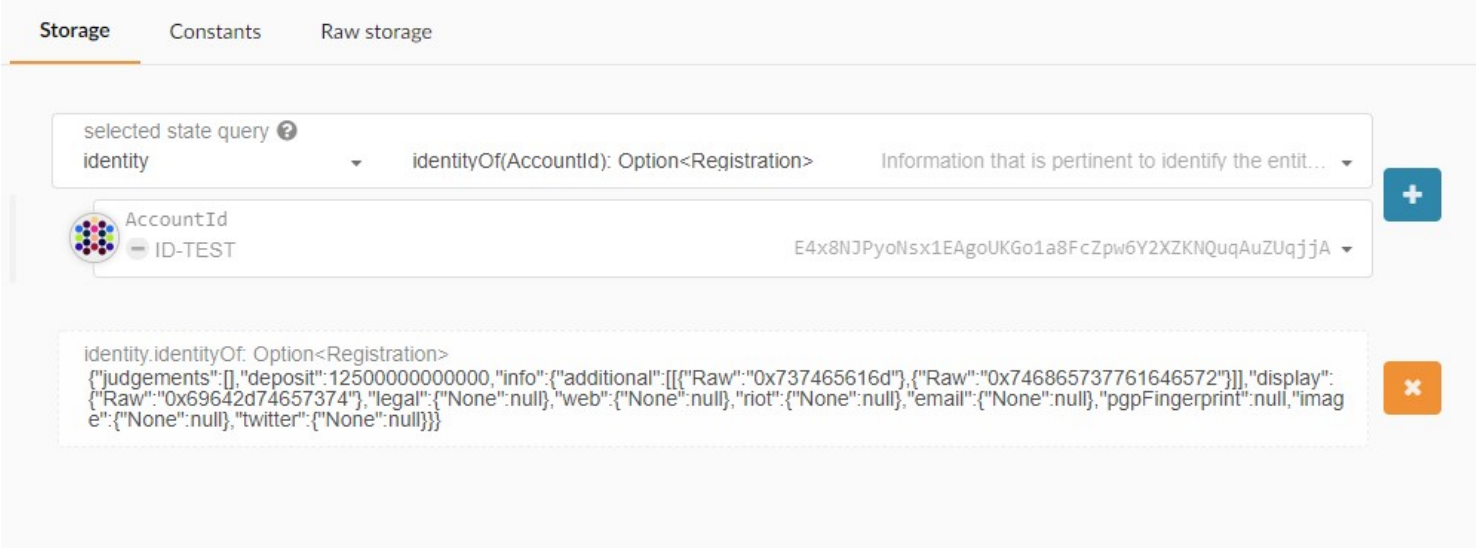
To add custom fields beyond the default ones, use the Extrinsics UI to submit a raw transaction by first clicking "Add Item" and adding any field name you like. The example below adds a field `steam`, which is a user's Steam username. The first value is the field name in bytes ("steam") and the second is the account name in bytes ("theswader"). The display name also has to be provided, otherwise, the Identity pallet would consider it wiped if we submitted it with the "None" option still selected. That is to say, every time you make a change to your identity values, you need to re-submit the entire set of fields: the write operation is always "overwrite", never "append".



Note that custom fields are not shown in the UI by default:



The rendering of such custom values is, ultimately, up to the UI/dapp makers. In the case of PolkadotJS, the team prefers to only show official fields for now. If you want to check that the values are still stored, use the Chain State UI to query the active account's identity info:



It is up to your own UI or dapp to then do with this data as it pleases. The data will remain available for querying via the Polkadot API, so you don't have to rely on the PolkadotJS UI.

You can have a maximum of 100 custom fields.

Format Caveat

Please note the following caveat: because the fields support different formats, from raw bytes to various hashes, a UI has no way of telling how to encode a given field it encounters. The PolkadotJS UI currently encodes the raw bytes it encounters as UTF8 strings, which makes these values readable on-screen. However, given that there are no restrictions on the values that can be placed into these fields, a different UI may interpret them as, for example, IPFS hashes or encoded bitmaps. This means any field stored as raw bytes will become unreadable by that specific UI. As field standards crystallize, things will become easier to use but for now, every custom implementation of displaying user information will likely have to make a conscious decision on the approach to take, or support multiple formats and then attempt multiple encodings until the output makes sense.

Registrars

Registrars can set a fee for their services and limit their attestation to certain fields. For example, a registrar could charge 1 DOT to verify one's legal name, email, and GPG key. When a user requests judgement, they will pay this fee to the registrar who provides the judgement on those claims. Users set a maximum fee they are willing to pay and only registrars below this amount would provide judgement.

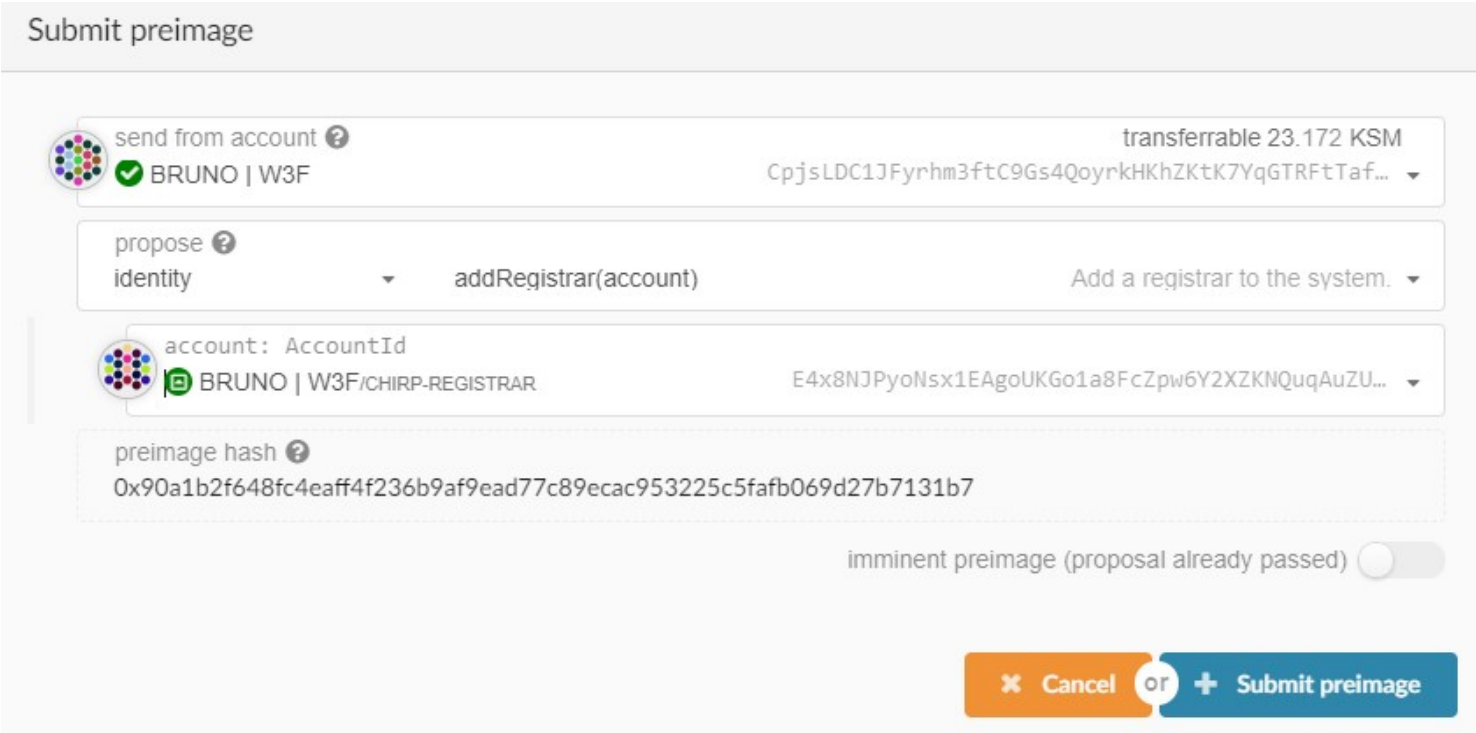
Becoming a registrar

To become a registrar, submit a pre-image and proposal into Democracy, then wait for people to vote on it. For best results, write a post about your identity and intentions beforehand, and once the

proposal is in the queue ask people to second it so that it gets ahead in the referendum queue.

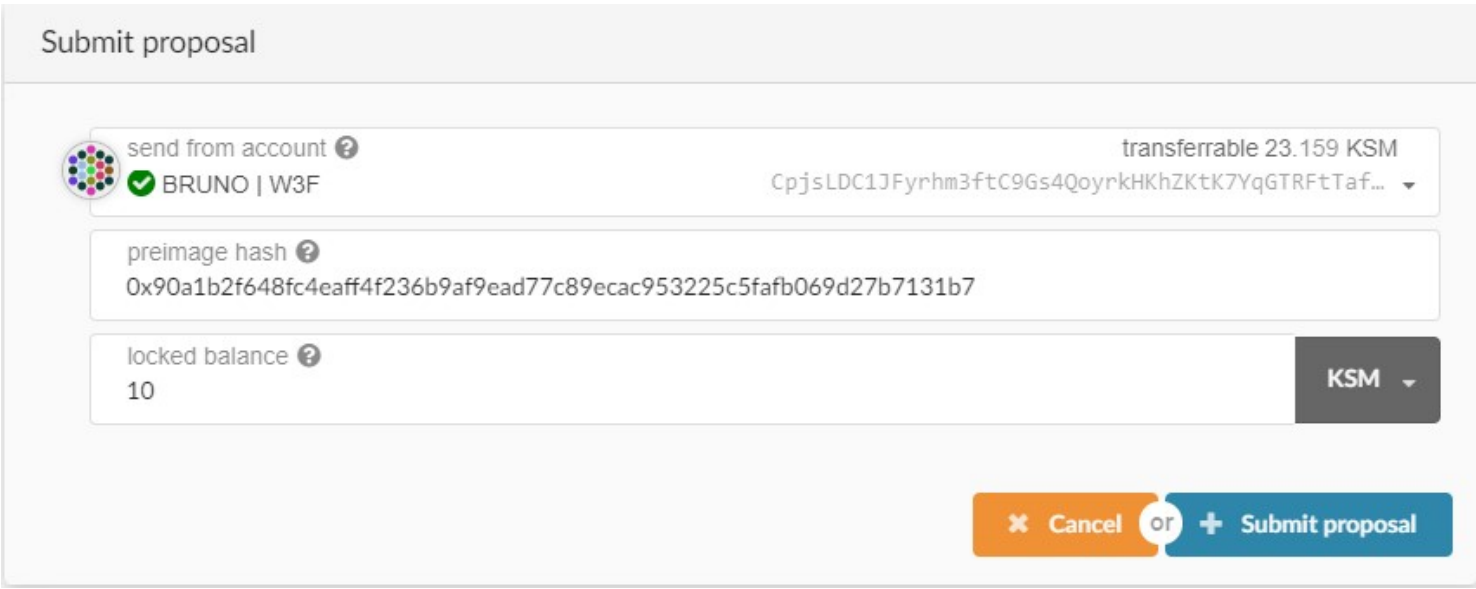
Here's how to submit a proposal to become a registrar:

Go to the Democracy tab, select "Submit preimage", and input the information for this motion - notably which account you're nominating to be a registrar in the `identity.setRegistrar` function.



Copy the preimage hash. In the above image, that's `0x90a1b2f648fc4eaff4f236b9af9ead77c89ecac953225c5fafb069d27b7131b7`. Submit the preimage by signing a transaction.

Next, select "Submit Proposal" and enter the previously copied preimage hash. The `locked balance` field needs to be at least {{ identity_reserve_funds }} KSM. You can find out the minimum by querying the chain state under Chain State -> Constants -> democracy -> minimumDeposit.



At this point, DOT holders can second the motion. With enough seconds, the motion will become a referendum, which is then voted on. If it passes, users will be able to request judgement from this registrar.

Judgements

After a user injects their information on chain, they can request judgement from a registrar. Users declare a maximum fee that they are willing to pay for judgement, and registrars whose fee is below that amount can provide a judgement.

When a registrar provides judgement, they can select up to six levels of confidence in their attestation:

- Unknown: The default value, no judgement made yet.
- Reasonable: The data appears reasonable, but no in-depth checks (e.g. formal KYC process) were performed.
- Known Good: The registrar has certified that the information is correct.
- Out of Date: The information used to be good, but is now out of date.
- Low Quality: The information is low quality or imprecise, but can be fixed with an update.
- Erroneous: The information is erroneous and may indicate malicious intent.

A seventh state, "fee paid", is for when a user has requested judgement and it is in progress. Information that is in this state or "erroneous" is "sticky" and cannot be modified; it can only be removed by complete removal of the identity.

Registrars gain trust by performing proper due diligence and would presumably be replaced for issuing faulty judgements.

To be judged after submitting your identity information, go to the "Extrinsics UI" and select the `identity` pallet, then `requestJudgement`. For the `reg_index` put the index of the registrar you want to be judged by, and for the `max_fee` put the maximum you're willing to pay for these confirmations.

If you don't know which registrar to pick, first check the available registrars by going to Chain State UI and selecting `identity.registrars()` to get the full list.

Cancelling Judgements

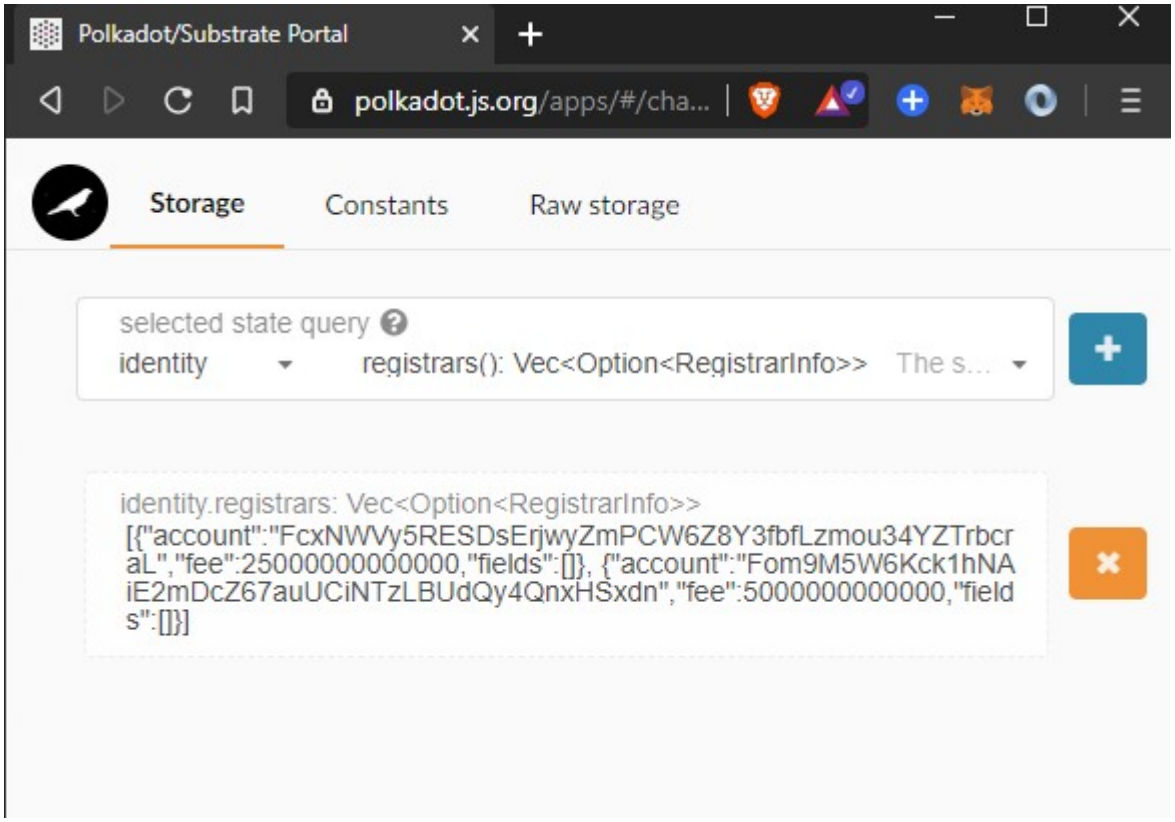
You may decide that you do not want to be judged by a registrar (for instance, because you realize you entered incorrect data or selected the wrong registrar). In this case, after submitting the request

for judgement but before your identity has been judged, you can issue a call to cancel the judgement using an extrinsic.

To do this, first, go to the "Extrinsics UI" and select the `identity` pallet, then `cancelRequest`. Ensure that you are calling this from the correct account (the one for which you initially requested judgement). For the `reg_index`, put the index of the registrar from which you requested judgement.

Submit the transaction, and the requested judgement will be cancelled.

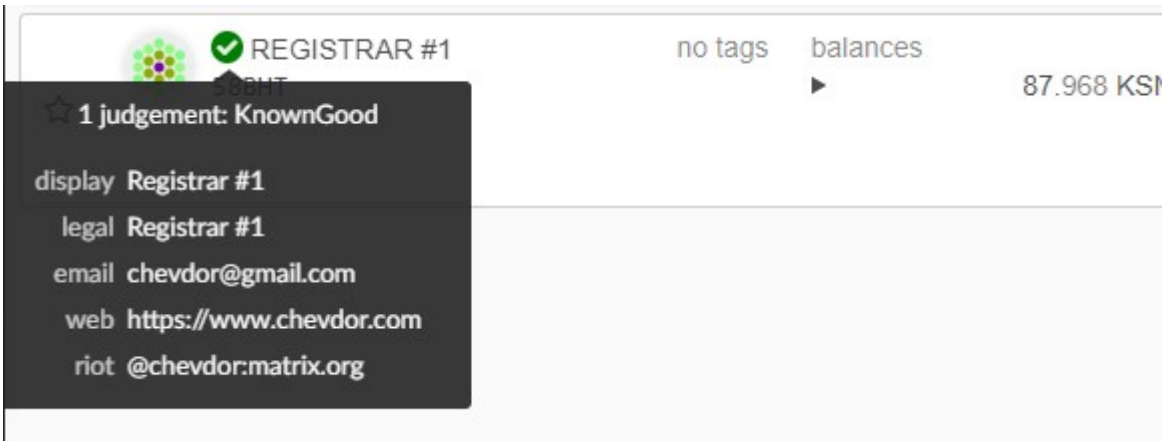
Kusama Registrars



The image above reveals there are two registrars on Kusama:

- Registrar 0, FcxNWVy5RESDsErjwyZmPCW6Z8Y3fbfLzmou34YZTrbcraL
- Registrar 1, Fom9M5W6Kck1hNAiE2mDcZ67auUCiNTzLBUdQy4QnxHSxdn

To find out how to contact the registrar after the application for judgement or to learn who they are, we can check their identity by adding them to our Address Book. Their identity will be automatically loaded.



Polkadot Registrars

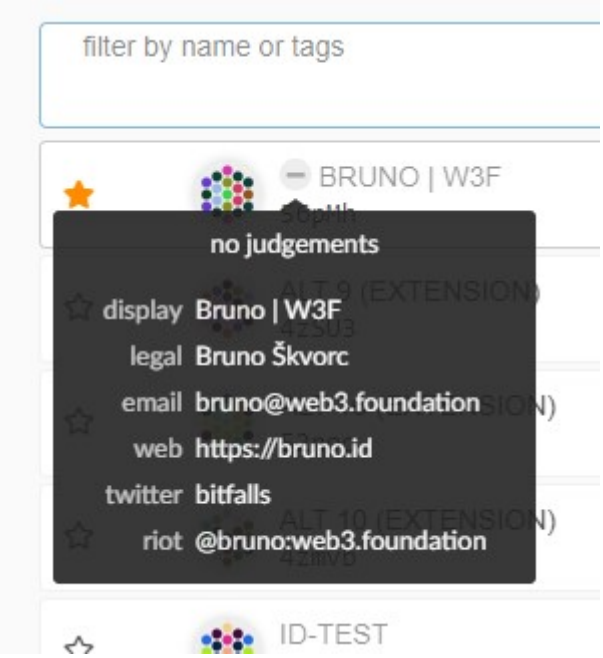
On Polkadot there are two registrars:

- Web3 Foundation is registrar #0. The W3F registrar service is available on the Kusama at the moment, follow the guide written here on how to use it. The service will be available on the Polkadot shortly.
- Chevdor is registrar #1.

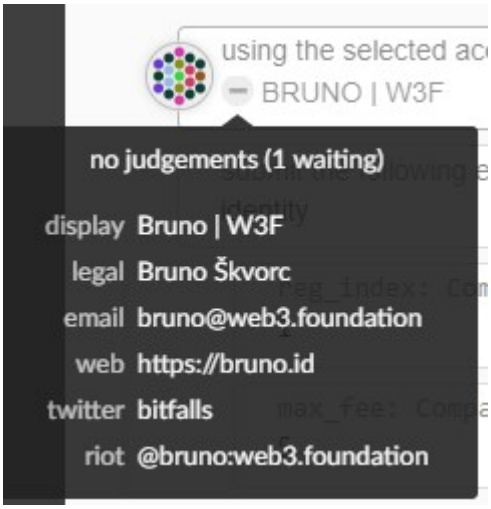
Requesting Judgement

Requesting judement follows the same process regardless of whether you're on the Kusama or Polkadot networks. Select one of the registrars from the query you made above.

This will make your identity go from unjudged:

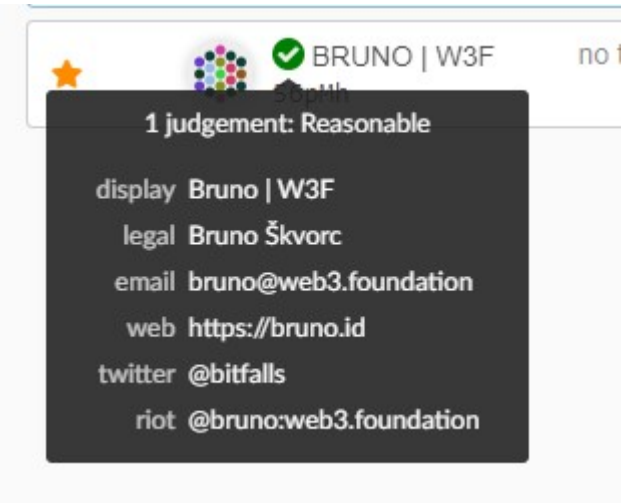


To "waiting":



At this point, direct contact with the registrar is required - the contact info is in their identity as shown above. Each registrar will have their own set of procedures to verify your identity and values, and only once you've satisfied their requirements will the process continue.

Once the registrar has confirmed the identity, a green checkmark should appear next to your account name with the appropriate confidence level:



Note that changing even a single field's value after you've been verified will un-verify your account and you will need to start the judgement process anew. However, you can still change fields while the judgement is going on - it's up to the registrar to keep an eye on the changes.

Sub Accounts

Users can also link accounts by setting "sub accounts", each with its own identity, under a primary account. The system reserves a bond for each sub account. An example of how you might use this would be a validation company running multiple validators. A single entity, "My Staking Company", could register multiple sub accounts that represent the Stash accounts of each of their validators.

An account can have a maximum of 100 sub-accounts.

To register a sub-account on an existing account, you must currently use the Extrinsics UI. There, select the identity pallet, then `setSubs` as the function to use. Click "Add Item" for every child account you want to add to the parent sender account. The value to put into the Data field of each parent is the optional name of the sub-account. If omitted, the sub-account will inherit the parent's name and be displayed as `parent/parent` instead of `parent/child`.

Note that a deposit of 2.5KSM is required for every sub-account.

Clearing and Killing an Identity

Clearing: Users can clear their identity information and have their deposit returned. Clearing an identity also clears all sub accounts and returns their deposits.

To clear an identity:

1. Navigate to the Accounts UI.

2. Click the three dots corresponding to the account you want to clear and select 'Set on-chain identity'.
3. Select 'Clear Identity', and sign and submit the transaction.

Killing: The Council can kill an identity that it deems erroneous. This results in a slash of the deposit.

Treasury

The Treasury is a pot of funds collected through transaction fees, slashing, staking inefficiencies, etc. The funds held in the Treasury can be spent by making a spending proposal that, if approved by the Council, will enter a waiting period before distribution. This waiting period is known as the budget period, and its duration is subject to governance, with the current default set to {{ spend_period }} days. The Treasury attempts to spend as many proposals in the queue as it can without running out of funds.

If the Treasury ends a budget period without spending all of its funds, it suffers a burn of a percentage of its funds -- thereby causing deflationary pressure. {{ polkadot: This percentage is currently at 1% on Polkadot. :polkadot }} {{ kusama: This percentage is currently 0.2% on Kusama, with the amount currently going to Society rather than being burned. :kusama }}

When a stakeholder wishes to propose a spend from the Treasury, they must reserve a deposit of at least 5% of the proposed spend (see below for variations). This deposit will be slashed if the proposal is rejected, and returned if it is accepted.

Proposals may consist of (but are not limited to):

- Infrastructure deployment and continued operation.
- Network security operations (monitoring services, continuous auditing).
- Ecosystem provisions (collaborations with friendly chains).
- Marketing activities (advertising, paid features, collaborations).
- Community events and outreach (meetups, pizza parties, hackerspaces).
- Software development (wallets and wallet integration, clients and client upgrades).

The Treasury is ultimately controlled by the Council, and how the funds will be spent is up to their judgment.

Funding the Treasury

The Treasury is funded from different sources:

1. Slashing: When a validator is slashed for any reason, the slashed amount is sent to the Treasury with a reward going to the entity that reported the validator (another validator). The reward is taken from the slash amount and varies per offence and number of reporters.

2. Transaction fees: A portion of each block's transaction fees goes to the Treasury, with the remainder going to the block author.
3. Staking inefficiency: Inflation is designed to be 10% in the first year, and the ideal staking ratio is set at 50%, meaning half of all tokens should be locked in staking. Any deviation from this ratio will cause a proportional amount of the inflation to go to the Treasury. In other words, if 50% of all tokens are staked, then 100% of the inflation goes to the validators as reward. If the staking rate is greater than or less than 50%, then the validators will receive less, with the remainder going to the Treasury.
4. Parathreads: Parathreads participate in a per-block auction for block inclusion. Part of this bid goes to the validator that accepts the block and the remainder goes to the Treasury.

Creating a Treasury Proposal

The proposer has to deposit 5% of the requested amount or {{ proposal_min_bond }} (whichever is higher) as an anti-spam measure. This amount is burned if the proposal is rejected, or refunded otherwise. These values are subject to governance so they may change in the future.

Please note that there is no way for a user to revoke a treasury proposal after it has been submitted. The Council will either accept or reject the proposal, and if the proposal is rejected, the bonded funds are burned.

Announcing the Proposal

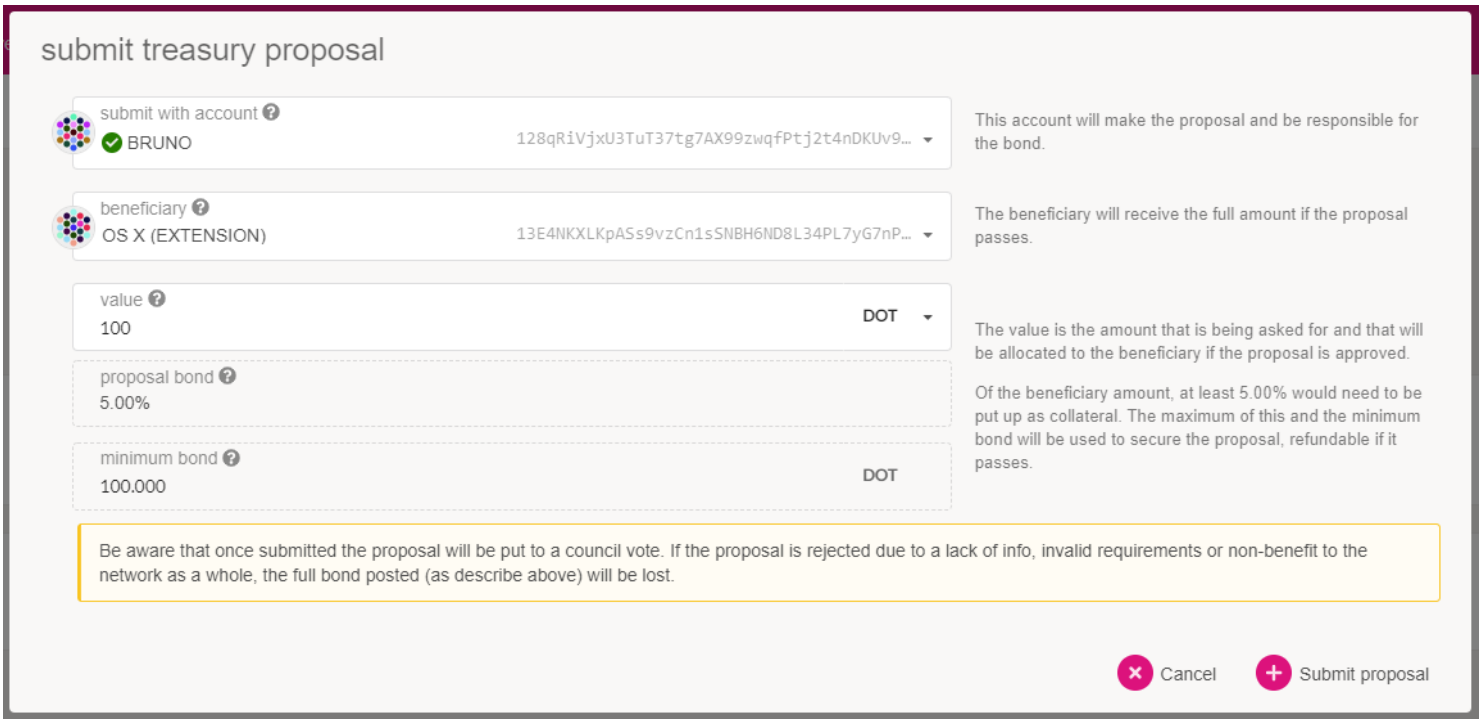
To minimize storage on chain, proposals don't contain contextual information. When a user submits a proposal, they will probably need to find an off-chain way to explain the proposal. Most discussion takes place on the following platforms:

- Many community members participate in discussion in the Kusama Element (previously Riot) chat or Polkadot Element.
- The Polkassembly discussion platform that allows users to log in with their Web3 address and automatically reads proposals from the chain, turning them into discussion threads. It also offers a sentiment gauge poll to get a feel for a proposal before committing to a vote.

Spreading the word about the proposal's explanation is ultimately up to the proposer - the recommended way is using official Element channels like {{ polkadot: the Polkadot Watercooler and Polkadot Direction room :polkadot }} {{ kusama: the Kusama Direction room or the Kusama Watercooler :kusama }}.

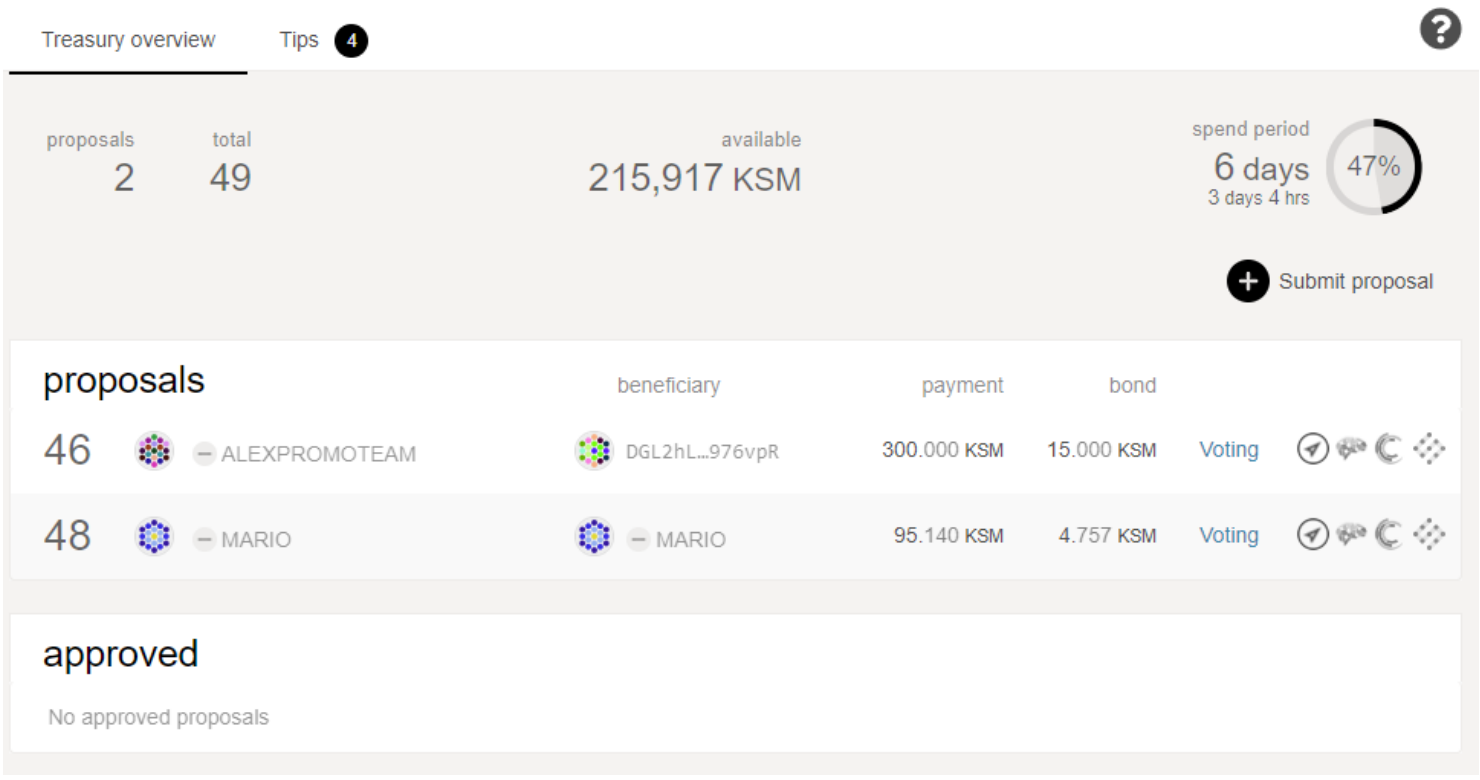
Creating the Proposal

One way to create the proposal is to use the Polkadot-JS Apps website. From the website, use either the extrinsics tab and select the Treasury pallet, then `proposeSpend` and enter the desired amount and recipient, or use the Treasury tab and its dedicated Submit Proposal button:



The system will automatically take the required deposit, picking the higher of the two values mentioned above.

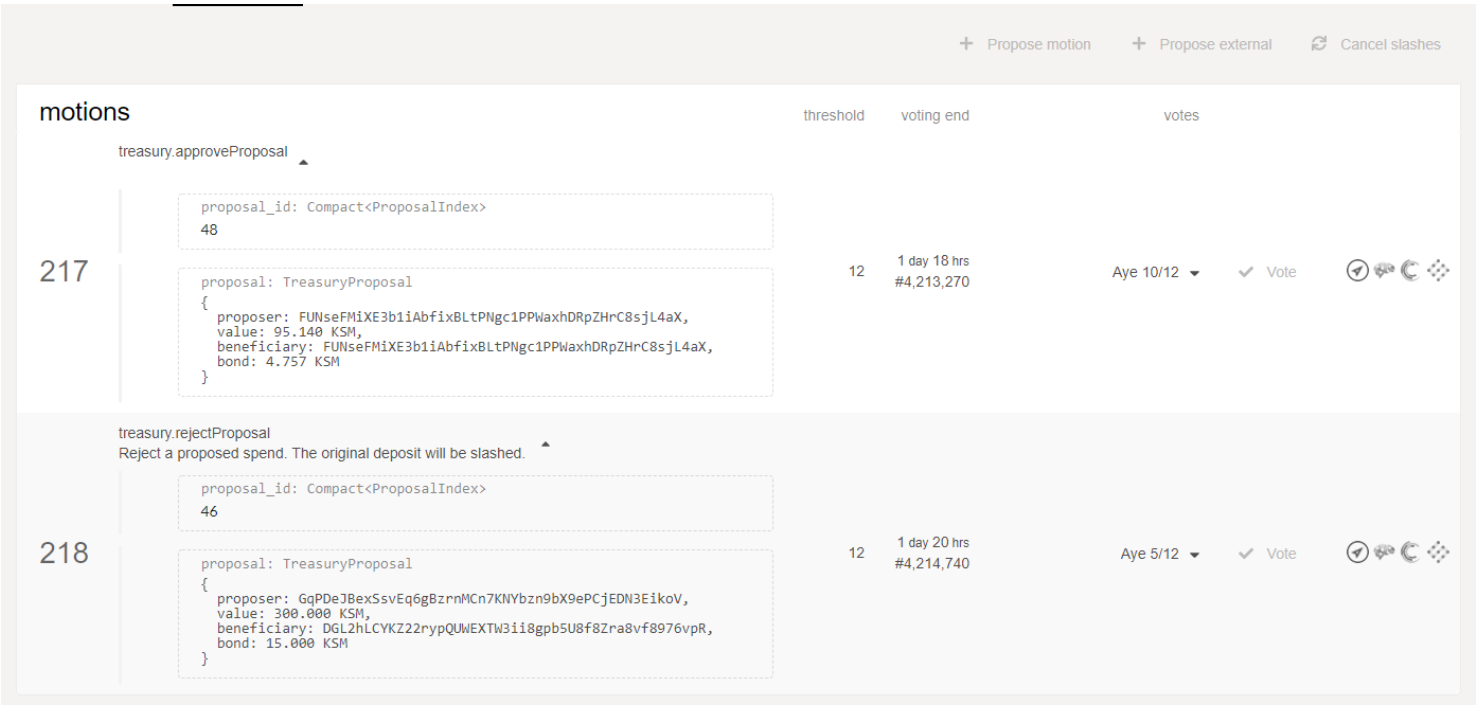
Once created, your proposal will become visible in the Treasury screen and the Council can start voting on it.



Remember that the proposal has no metadata, so it's up to the proposer to create a description and purpose that the Council could study and base their votes on.

At this point, a Council member can create a motion to accept or to reject the treasury proposal. It is possible that one motion to accept and another motion to reject are both created. The proportions to accept and reject Council proposals vary between accept or reject, and possibly depend on which network the Treasury is implemented.

The threshold for accepting a treasury proposal is at least three-fifths of the Council. On the other hand, the threshold for rejecting a proposal is at least one-half of the Council.



Tipping

Next to the proposals process, a separate system for making tips exists for the Treasury. Tips can be suggested by anyone and are supported by members of the Council. Tips do not have any definite value; the final value of the tip is decided based on the median of all tips issued by the tippers.

Currently, the tippers are the same as the members of the Council. However, being a tipper is not the direct responsibility of the Council, and at some point the Council and the tippers may be different groups of accounts.

A tip will enter a closing phase when more than a half plus one of the tipping group have endorsed a tip. During that timeframe, the other members of the tipping group can still issue their tips, but do not have to. Once the window closes, anyone can call the `close_tip` extrinsic, and the tip will be paid out.

There are two types of tips: public and tipper-initiated. With public tips, a small bond is required to place them. This bond depends on the tip message length, and a fixed bond constant defined on chain, currently {{ tip_deposit_amount }}. Public tips carry a finder's fee of {{ tip_finders_fee }}%

which is paid out from the total amount. Tipper-initiated tips, i.e. tips that a Council member published, do not have a finder's fee or a bond.

To better understand the process a tip goes through until it is paid out, let's consider an example.

Example

Bob has done something great for {{ polkadot: Polkadot :polkadot }} {{ kusama: Kusama :kusama }}. Alice has noticed this and decides to report Bob as deserving a tip from the Treasury. The Council is composed of three members Charlie, Dave, and Eve.

Alice begins the process by issuing the `report_awesome` extrinsic. This extrinsic requires two arguments, a reason and the address to tip. Alice submits Bob's address with the reason being a UTF-8 encoded URL to a post on {{ polkadot: polkassembly :polkadot }} {{ kusama: polkassembly :kusama }} that explains her reasoning for why Bob deserves the tip.

As mentioned above, Alice must also lock up a deposit for making this report. The deposit is the base deposit as set in the chain's parameter list plus the additional deposit per byte contained in the reason. This is why Alice submitted a URL as the reason instead of the explanation directly, it was cheaper for her to do so.

For her trouble, Alice is able to claim the eventual finder's fee if the tip is approved by the tippers.

Since the tipper group is the same as the Council, the Council must now collectively (but also independently) decide on the value of the tip that Bob deserves.

Charlie, Dave, and Eve all review the report and make tips according to their personal valuation of the benefit Bob has provided to Kusama.

Charlie tips {{ polkadot: 10 DOT :polkadot }} {{ kusama: 1 KSM :kusama }}. Dave tips {{ polkadot: 30 DOT :polkadot }} {{ kusama: 3 KSM :kusama }}. Eve tips {{ polkadot: 100 DOT :polkadot }} {{ kusama: 10 KSM :kusama }}.

The tip could have been closed out with only two of the three tippers. Once more than half of the tippers group have issued tip valuations, the countdown to close the tip will begin. In this case, the third tipper issued their tip before the end of the closing period, so all three were able to make their tip valuations known.

Now the actual tip that will be paid out to Bob is the median of these tips, so Bob will be paid out {{ polkadot: 30 DOT :polkadot }} {{ kusama: 3 KSM :kusama }} from the Treasury.

In order for Bob to be paid his tip, some account must call the `close_tip` extrinsic at the end of the closing period for the tip. This extrinsic may be called by anyone.

Bounties Spending

There are practical limits to Council Members curation capabilities when it comes to treasury proposals: Council members likely do not have the expertise to make a proper assessment of the activities described in all proposals. Even if individual Councillors have that expertise, it is highly unlikely that a majority of members are capable in such diverse topics.

Bounties Spending proposals aim to delegate the curation activity of spending proposals to experts called Curators: They can be defined as addresses with agency over a portion of the Treasury with the goal of fixing a bug or vulnerability, developing a strategy, or monitoring a set of tasks related to a specific topic: all for the benefit of the Polkadot ecosystem.

A proposer can submit a bounty proposal for the Council to pass, with a curator to be defined later, whose background and expertise is such that they are capable of determining when the task is complete. Curators are selected by the Council after the bounty proposal passes, and need to add an upfront payment to take the position. This deposit can be used to punish them if they act maliciously. However, if they are successful in their task of getting someone to complete the bounty work, they will receive their deposit back and part of the bounty reward.

When submitting the value of the bounty, the proposer includes a reward for curators willing to invest their time and expertise in the task: this amount is included in the total value of the bounty. In this sense, the curator's fee can be defined as the result of subtracting the value paid to the bounty rewardee from the total value of the bounty.

In general terms, curators are expected to have a well-balanced track record related to the issues the bounty tries to resolve: they should be at least knowledgeable on the topics the bounty touches, and show project management skills or experience. These recommendations ensure an effective use of the mechanism. A Bounty Spending is a reward for a specified body of work - or specified set of objectives - that needs to be executed for a predefined treasury amount to be paid out. The responsibility of assigning a payout address once the specified set of objectives is completed is delegated to the curator.

After the Council has activated a bounty, it delegates the work that requires expertise to the curator who gets to close the active bounty. Closing the active bounty enacts a delayed payout to the payout address and a payout of the curator fee. The delay phase allows the Council to act if any issues arise.

To minimize storage on chain in the same way as any proposal, bounties don't contain contextual information. When a user submits a bounty spending proposal, they will probably need to find an off-chain way to explain the proposal (any of the available community forums serve this purpose). This template can help as a checklist of all needed information for the Council to make an informed decision.

The bounty has a predetermined duration of 90 days with the possibility of being extended by the curator. Aiming to maintain flexibility on the tasks' curation, the curator will be able to create sub-bounties for more granularity and allocation in the next iteration of the mechanism.

Creating a Bounty Proposal

Anyone can create a Bounty proposal using Polkadot JS Apps: Users are able to submit a proposal on the dedicated Bounty section under Governance. The development of a robust user interface to view and manage bounties in the Polkadot Apps is still under development and it will serve Council members, Curators and Beneficiaries of the bounties, as well as all users observing the on-chain treasury governance. For now, the help of a Councillor is needed to open a bounty proposal as a motion to be voted.

To submit a bounty, please visit Polkadot JS Apps and click on the governance tab in the options bar on the top of the site. After, click on 'Bounties' and find the button '+ Add Bounty' on the upper-right side of the interface. Complete the bounty title, the requested allocation (including curator's fee) and confirm the call.

After this, a Council member will need to assist you to pass the bounty proposal for vote as a motion. You can contact the Council by joining the {{ polkadot: Polkadot Direction channel :polkadot }} {{ kusama: Kusama Direction channel :kusama }} in Element or joining our {{ polkadot: Polkadot Discord server :polkadot }} {{ kusama: Kusama Discord server :kusama }} and publishing a short description of your bounty, with a link to one of the forums for contextual information.

A bounty can be cancelled by deleting the earmark for a specific treasury amount or be closed if the tasks have been completed. On the opposite side, the 90 days life of a bounty can be extended by amending the expiry block number of the bounty to stay active.

Closing a bounty

The curator can close the bounty once they approve the completion of its tasks. The curator should make sure to set up the payout address on the active bounty beforehand. Closing the Active bounty enacts a delayed payout to the payout address and a payout of the curator fee.

A bounty can be closed by using the extrinsics tab and selecting the Treasury pallet, then `Award_bounty`, making sure the right bounty is to be closed and finally sign the transaction. It is important to note that those who received a reward after the bounty is completed, must claim the specific amount of the payout from the payout address, by calling `Claim_bounty` after the curator closed the allocation.

To understand more about Bounties and how this new mechanism works, read this Polkadot Blog post.

FAQ

What prevents the Treasury from being captured by a majority of the Council?

The majority of the Council can decide the outcome of a treasury spend proposal. In an adversarial mindset, we may consider the possibility that the Council may at some point go rogue and attempt to steal all of the treasury funds. It is a possibility that the treasury pot becomes so great, that a large financial incentive would present itself.

For one, the Treasury has deflationary pressure due to the burn that is suffered every spend period. The burn aims to incentivize the complete spend of all treasury funds at every burn period, so ideally the treasury pot doesn't have time to accumulate mass amounts of wealth. However, it is the case that the burn on the Treasury could be so little that it does not matter - as is the case currently on Kusama with a 0.2% burn.

However, it is the case on Kusama that the Council is composed of mainly well-known members of the community. Remember, the Council is voted in by the token holders, so they must do some campaigning or otherwise be recognized to earn votes. In the scenario of an attack, the Council members would lose their social credibility. Furthermore, members of the Council are usually externally motivated by the proper operation of the chain. This external motivation is either because they run businesses that depend on the chain, or they have direct financial gain (through their holdings) of the token value remaining steady.

Concretely, there are a couple on-chain methods that resist this kind of attack. One, the Council majority may not be the token majority of the chain. This means that the token majority could vote to replace the Council if they attempted this attack - or even reverse the treasury spend. They would do this through a normal referendum. Two, there are time delays to treasury spends. They are only enacted every spend period. This means that there will be some time to observe this attack is taking place. The time delay then allows chain participants time to respond. The response may take the form of governance measures or - in the most extreme cases a liquidation of their holdings and a migration to a minority fork. However, the possibility of this scenario is quite low.

Further Reading

- Substrate's Treasury Pallet - The Rust implementation of the Treasury. (Docs)

Using W3F Registrar

An on-chain identity is a good way to build up your reputation and let the community know more about you if you plan on running a validator or being a councilor. Web3 Foundation provides a registrar service in the Kusama and Polkadot networks that only charges a small fee (0.04 KSM) on Kusama, and no fees on Polkadot (although there may be a small fee in the future). However, you will of course need to reserve some DOT in your account while you have an identity, no matter which registrar you use. For details on amount necessary to reserve, as well as the identity system as a whole, see the identity page.

> **Note**: The registrar bot will not ask you to send any DOT, and never expose your private keys to anyone!

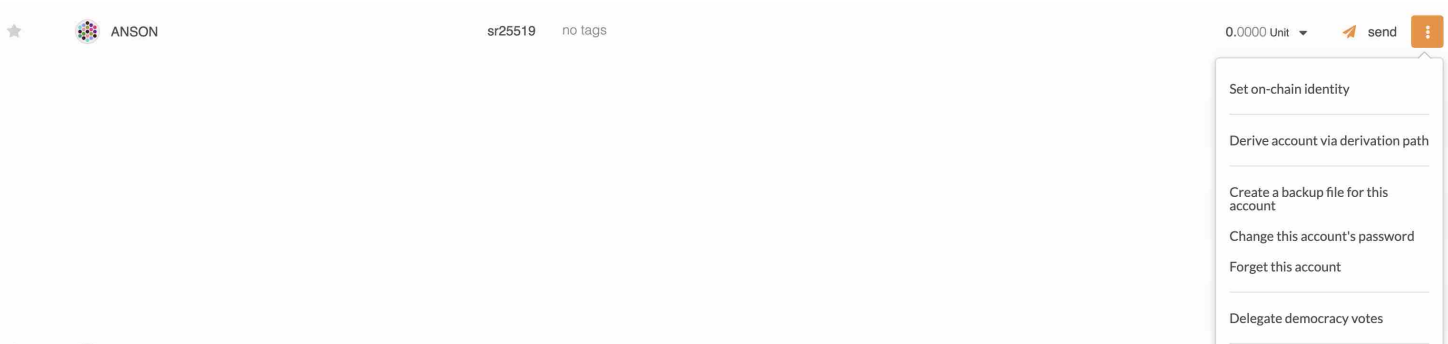
If you have provided `display name`, `email`, `twitter`, or `element name (previously called Riot)` when setting an on-chain identity, these will be required to verify one by one by signing a challenge message. Be aware that the `display name` cannot be too similar to others that have verified already. There is no need to set all of the fields when using the service. You are free to set whichever field or fields that you like.

Free feel to join the Polkadot's community to ask questions if there is anything unclear.

Setting an On-chain Identity

> **Note**: The W3F Registrar currently **does not** support KYC or web verification. Make sure to leave it blank when you fill in your identity information.

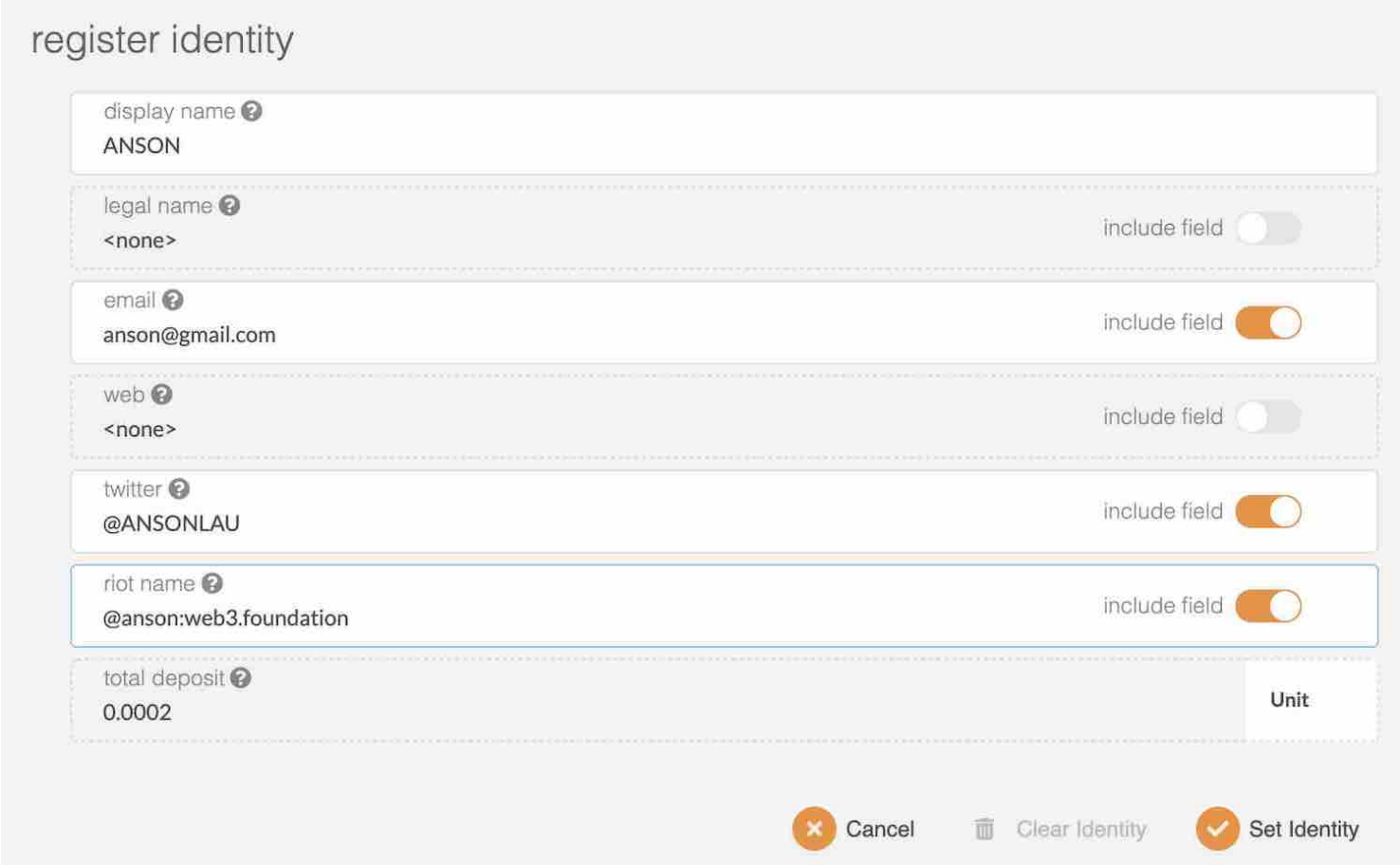
Go to Accounts page in Polkadot-JS Apps. The easiest way to add the built-in fields is to click the vertical three dots next to one's account and select "Set on-chain identity".



A popup will appear, offering the default fields.

Currently, the registrar only supports the following fields:

- Display Name.
- Element (formerly known as Riot)
- Email
- Twitter



Once you have filled in the information you would like to store on-chain, click `Set Identity` to submit the transaction.

Now you have set the identity information on-chain, but that is not verified yet, so you should see a little grey icon beside your name. It is the time to interact with the W3F's verification bot by submitting the judgment request to the W3F's registrar.

Request Judgement

Go to Developer->Extrinics and select your account to submit the `identity -> requestJudgement(reg_index, max_fee)` transaction. This will request the registrar to validate the information you set on-chain earlier.

The `reg_index` is the position of the registrar. For W3F, use 0.

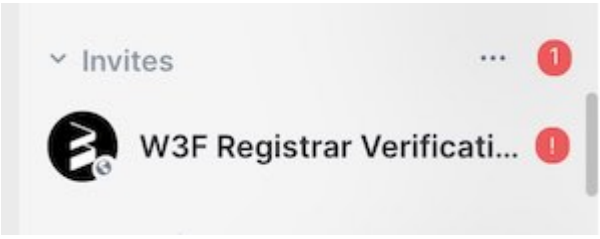
The `max_fee` is the amount of DOT or KSM to pay the registrar. For Kusama use **0.04 KSM** and for Polkadot use **0 DOT**.

Note that in the future, a fee may be charged for the Polkadot registrar.

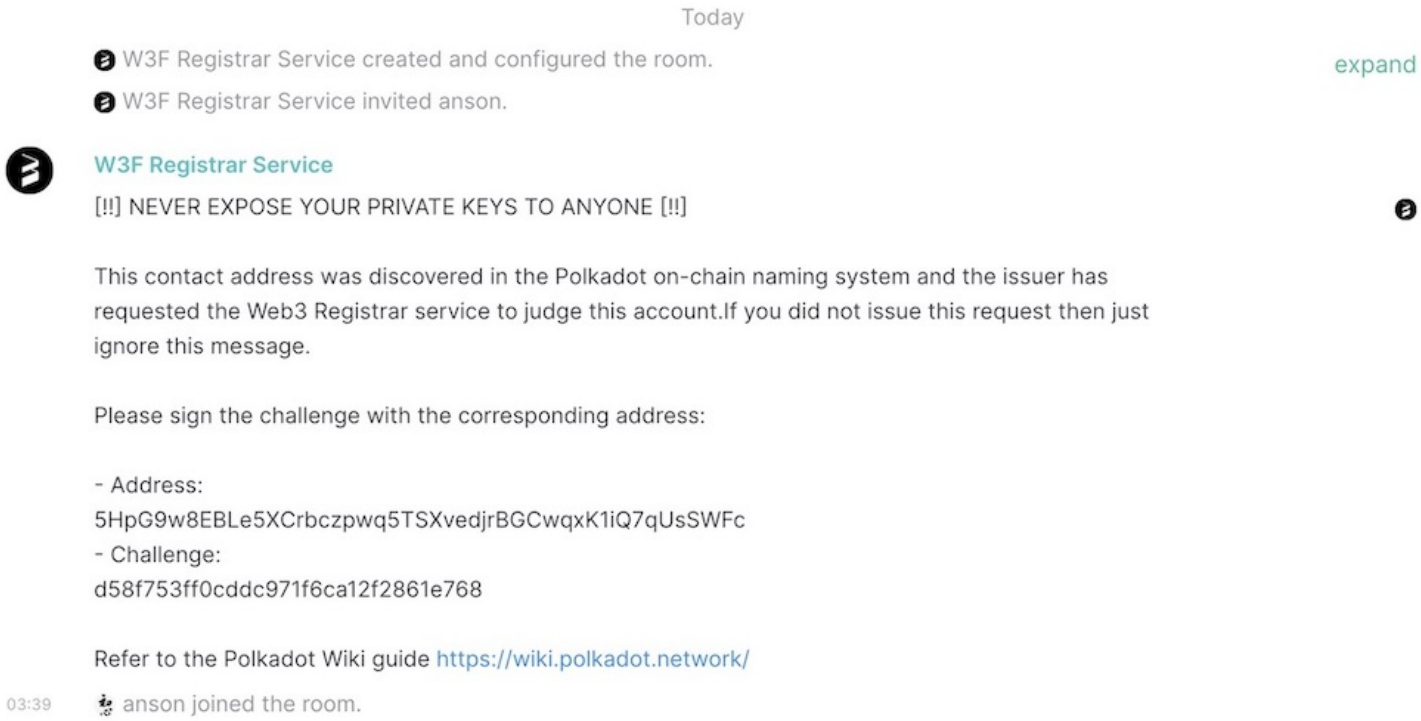
Element Verification

Since we provided the Element, Twitter, and Email information in this example, we would start to receive the verification requests from those platforms. As for Element, an invitation will be sent by the bot named "W3F Registrar Verification".

> Note: The handle of the W3F bot is called @registrar:web3.foundation. If you are not sure whether that is ours or not, ask in the Polkadot community chat first.



Once you accept the invitation, you should see the following information.



Then go to Sign and Verify under the Developer tab in the PolkadotJS and select your account, paste the "Challenge" data to the "sign the following data" field and click "Sign message".

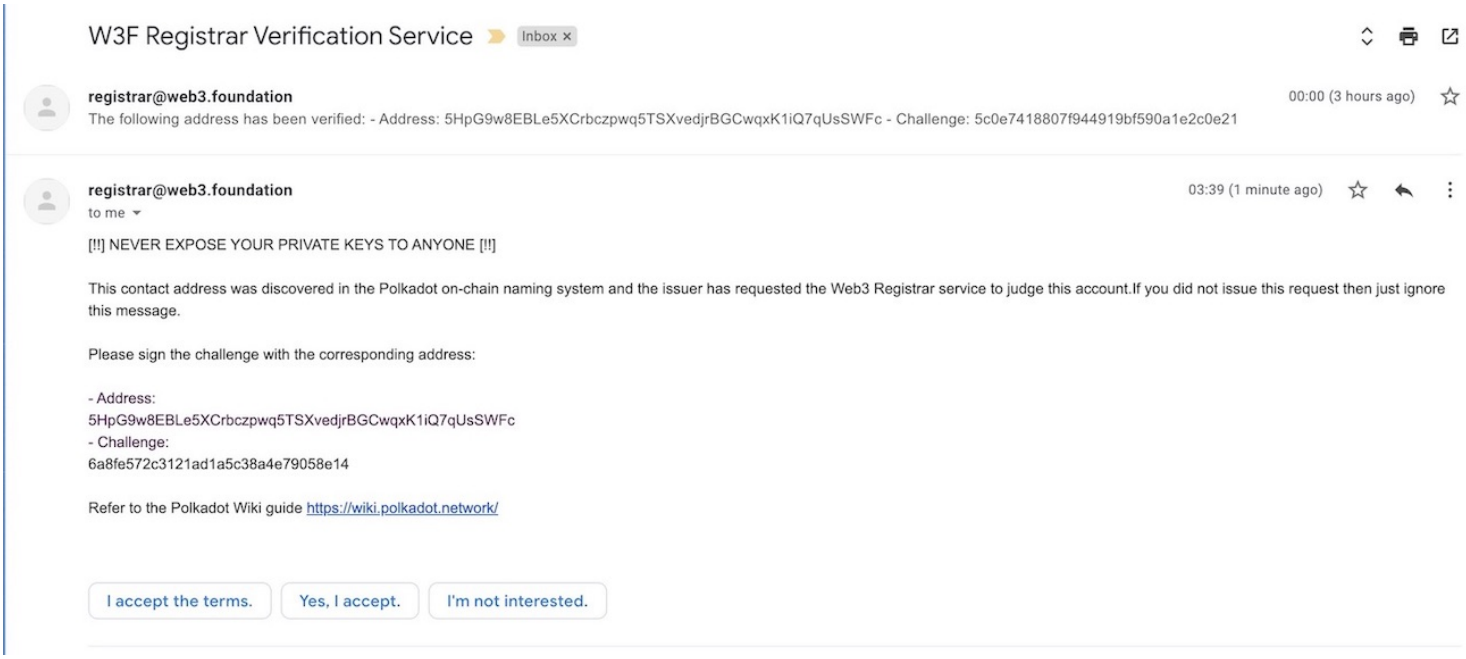
Copy the result of the "signature of supplied data" and paste it to the W3F Registrar Verification chat.

If the information is correct, you should see a message like the above image that indicates your address has been verified. This basically proves you are the owner of the account.

Email Verification

Next, you should receive an email called "W3F Registrar Verification Service". Below is an example for reference.

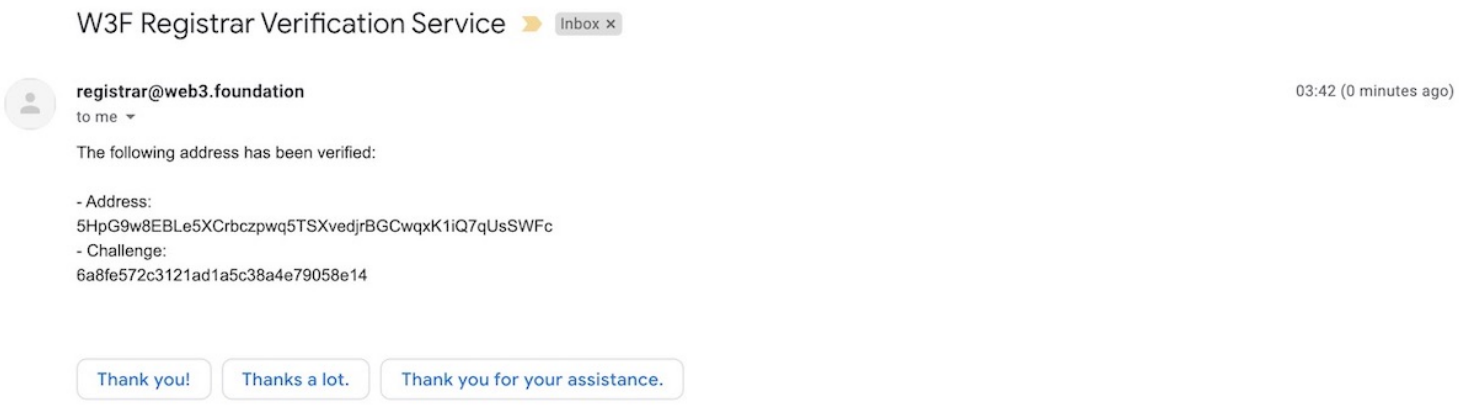
> Note: Please double-check the sender is "registrar@web3.foundation", not the others.



You would do what you did in the above again. Copy the "Challenge" data and go to Sign and Verify under the Developer tab in the PolkadotJS and select your account, paste the "Challenge" data to the "sign the following data" field and click "Sign message"

And reply with your signed data only in the email. Then click "Send".

> Note: Do not add anything in the email except the signed data.



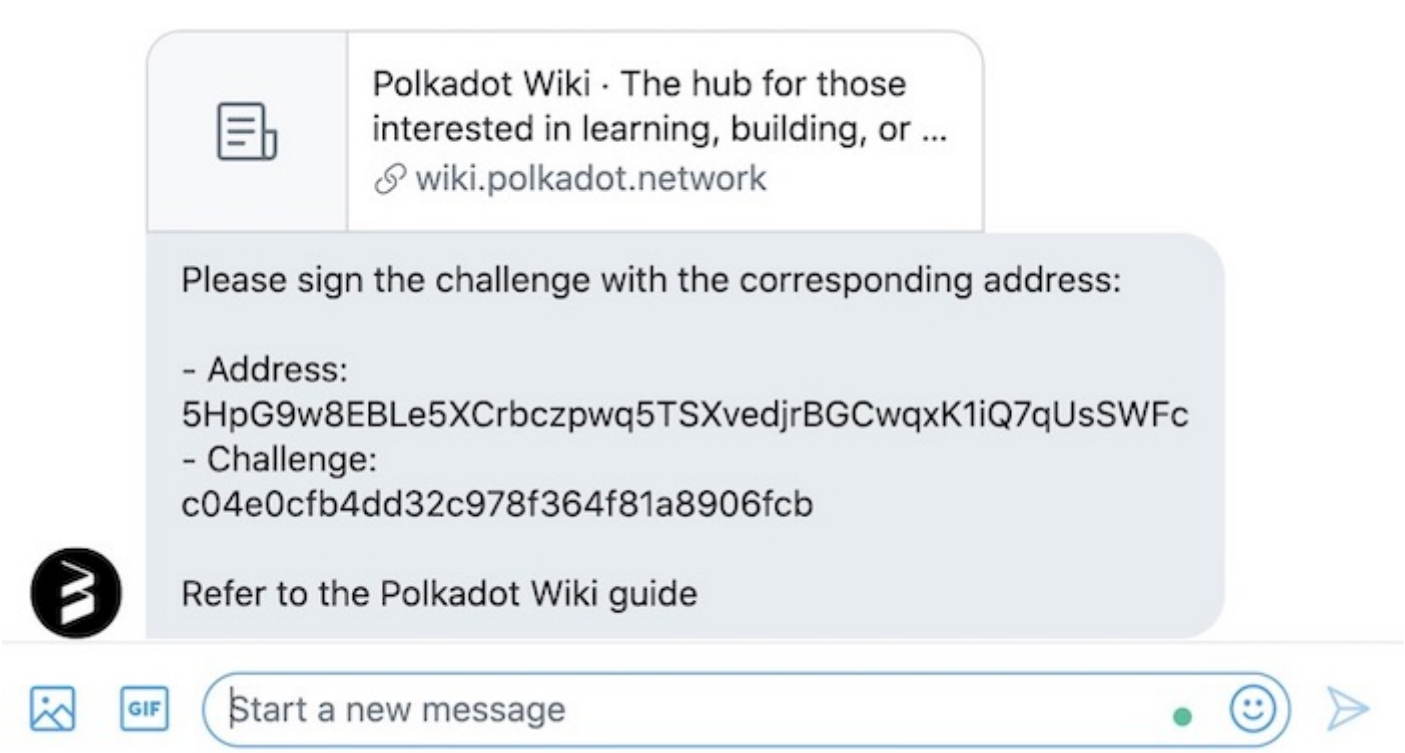
Wait 1 or 2 minutes. You should receive another email that shows your email has been verified successfully.

Twitter Verification

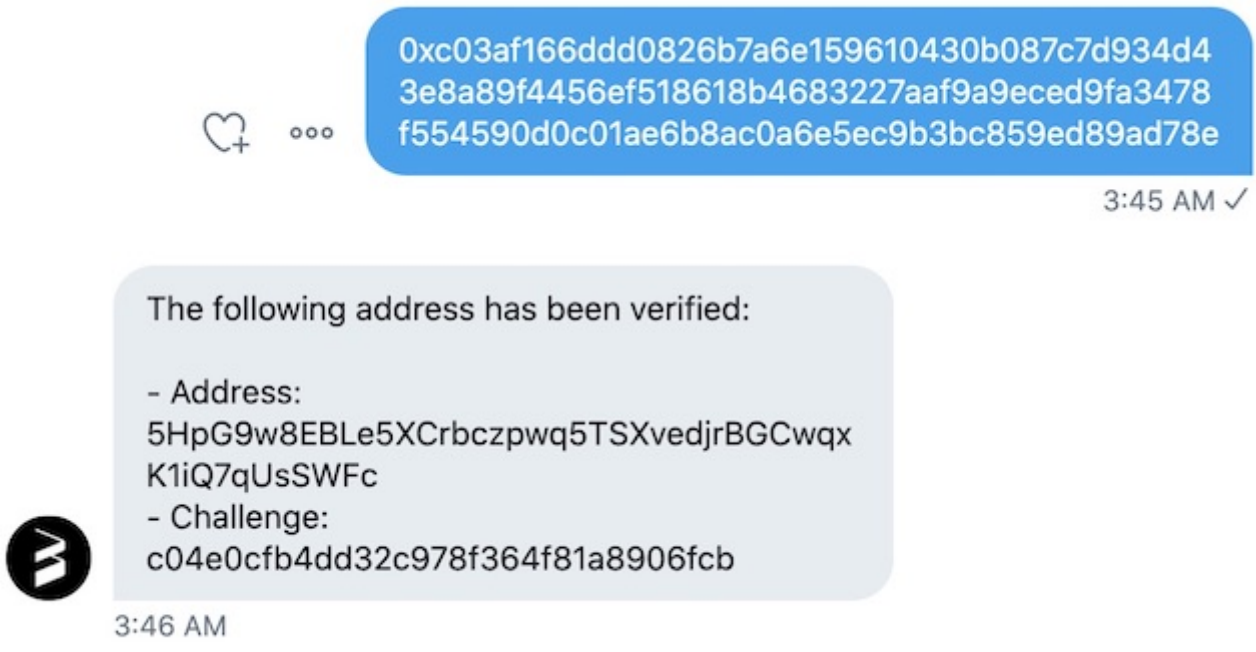
Lastly, if you have provided Twitter handle, you would have to follow @w3f_registrar first.

After following the Registrar account on Twitter, you will need to send it a DM. A simple "hello" will do the trick.

After waiting a few mintues you should receive a challenge similar to the previous two.



Again, just like how you did in the above. By using your account to sign the "Challenge" data that you received on Twitter in the Sign and Verify page.



Paste the signed data to the chat and you would receive the verification status after 1 to 2 minutes.

If everything has been verified successfully, you would see your account verification status has been marked as "reasonable" with a green tick icon on the Accounts page.

Congratulations! Your identity should now show as a green "verified" checkmark on Polkadot-JS Apps.

Polkadot Builders Starter's Guide

This article is the maintained version of the blog post: Everything you Need to Know to Prepare for Polkadot.

Polkadot is a blockchain protocol with two goals: providing **shared security** among all connected parachains and allowing all connected chains to **interoperate** by using XCMP. With the advent of PDKs like Parity Substrate and Cumulus, the time it takes to develop and launch a new chain has dropped significantly. While before it would take years to launch a new chain, now it may only take weeks or even days.

This guide will walk you through the steps you can take today to get started building your vision with Polkadot. It will explain the difference between a parachain and a smart contract (and why one may be better suited for your application over the other).

Polkadot Ecosystem Networks

- Mainnet: **Polkadot**
- Canary network: **Kusama**
  - Kusama is a value-bearing canary network that gets features before Polkadot does... *expect chaos*...
- Official testnets:
  - **Westend** - Functionality equal to the current Polkadot mainnet, with possible next-generation testing of features from time to time that will eventually migrate onto Polkadot.
  - **Canvas** - Wasm based Smart Contract enabled testnet, primarily for `ink!` development.
  - **Rococo** - Parachains and XCM testnet.

Polkadot mainnet has been running since May 2020 and has implementations in various programming languages ranging from Rust to JavaScript. Currently, the leading implementation is built in Rust and built using the Substrate framework.

Tooling is rapidly evolving to interact with the network; there are so many ways to get started!

But before you jump head-first into the code, you should consider the *kind* of decentralized application you want to make and understand the different paradigms available to developers who want to build on Polkadot.

What is the difference between building a parachain, a parathread, or a smart contract?

Polkadot provides a few ways for you to deploy your application: as a smart contract on an existing parachain, as your own parachain, or as a parathread. There are trade-offs when working with each of these and reading this section will help you understand them.

Parachains & Parathreads

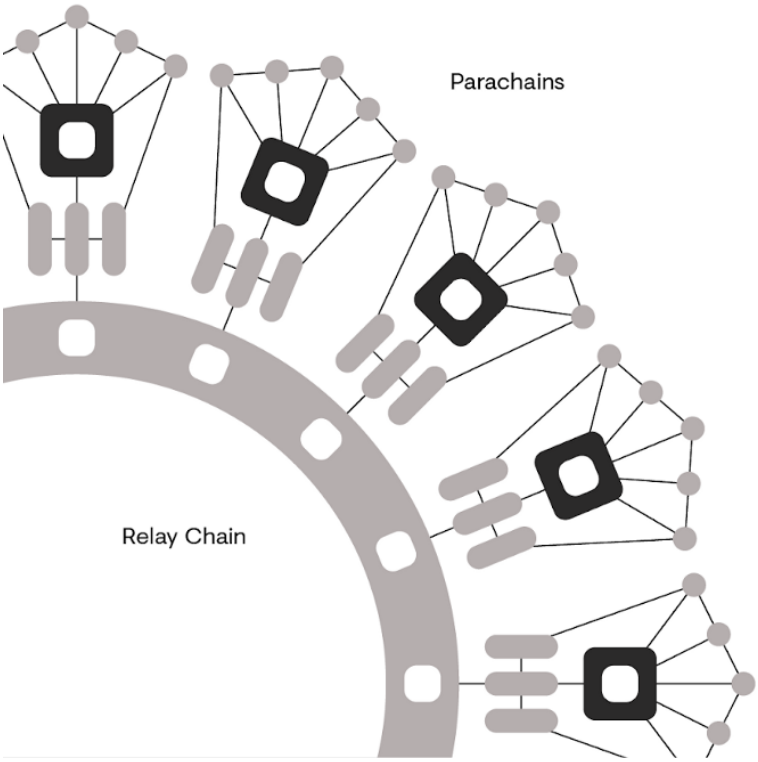
Parachains are "parallel" chains containing their own runtime logic and can benefit from the shared security and the cross-chain messaging provided by the Polkadot Relay Chain. Parachains permit a high degree of flexibility and customization but require more effort to create and maintain over time.

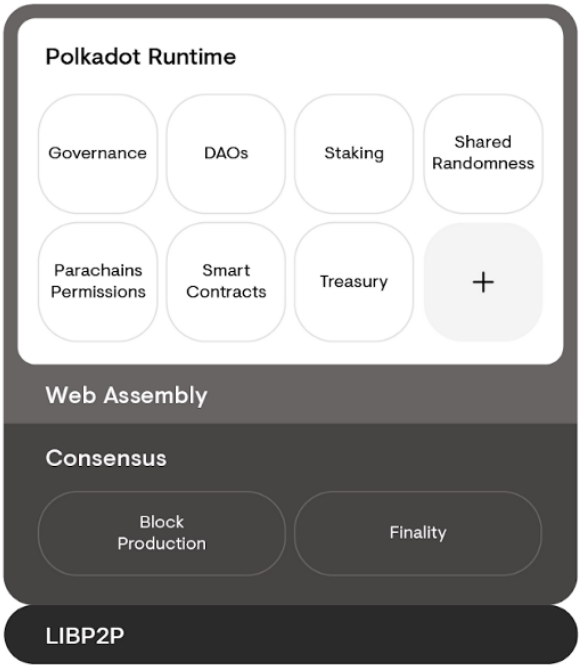
Parathreads are like parachains and enable the developer to have lower-level control of the logic of their application. The main difference between the two is economic since parathreads will be much less expensive to secure than a parachain. The lower costs of parathreads are due to the fact that parathreads will only produce a block when they need to, unlike parachains, which have secured a slot to produce a block at every block of the Relay Chain. When building a parathread, you will use the same tools (like PDKs) and you get all of the benefits of building a parachain, without the drawback of the cost.

Parachains grant the creators more space to build the monetary system and other aspects of the chain from the ground up. They will allow for more succinct and efficient execution of complex logic than could ever be offered by a smart contract platform. Parachains also offer more flexibility in the form of governance and can perform complete upgrades in a less controversial way than the current process of hard-forks.

Some examples of features you can have on a parachain or parathread:

- Custom fee structure (for example, pay a flat fee for transactions or pay per byte).
- Custom monetary policy for the native token and local economy.
- Treasury to be funded through transitions in your state function.
- A governance mechanism that could manage a DAO that is responsible for allocating your on-chain treasury.





Parachains open possibilities to construct complex runtime logic that would be too expensive to execute with smart contracts. However, unlike smart contracts, parachains lack a mandatory gas metering system entirely and could potentially be vulnerable to bugs that cause infinite loops (something that is prevented by design in smart contracts). This vulnerability is mitigated by the weight system that is implemented in Substrate -- although it places more of a burden on the developer of the parachain to properly perform benchmarks.

You may also decide to harness a combination of parachain, parathread, and smart contract. If you have certain logic that requires loops and it cannot be removed, use the native parachain runtime to handle all complex logic and the smart contract to call iteration. If you require off-chain data from an oracle, you may want to use a parathread as an oracle feed that only triggers once every 24 hours (this makes the most sense if the data is useful to other players in the Polkadot ecosystem too).

Most likely you've already realized that your application is better suited to be one or the other (or a hybrid of them both), but if you need a quick recap to digest the information, you can use this comparison chart as a cheat sheet:

	Parachain	Smart Contract
Speed of development	–	+
Ease of deployment	–	+
Complexity of logic	+	–
Maintainence overhead	–	+
Level of customization	+	–
Strict resource control	–	+
Native chain features	+	–
Scalability	+	–

> **Note:** The image above does not include parathreads, but as we mentioned before all the benefits of parachains apply just as well to parathreads. Parathreads, however, *are* cheaper to deploy and maintain. So if they had a column on the table above, it would look like the parachain column with "Ease of deployment" and "Maintenance overhead" changed to `+`.

Smart Contracts

A smart contract is simply some code that exists at an address on a chain and is callable by external actors. The key part is that you actually have to put the code on chain before anyone can start executing it!

Deploying your smart contract on chain will vary slightly for whichever specific parachain you will use, but generally you will send a special transaction that will create the smart contract on the ledger. You will likely need to pay an associated fee for the initialization logic and any storage that your contract consumes.

On the Polkadot mainnet, there will be parachains that act as smart contract platforms. Smart contracts are executable programs that exist on only a single chain and are limited in complexity. Because they exist on a single chain, they can have smooth interoperability with other smart contracts on the same chain. However, they will always be constrained and limited by the inherent characteristics of their host chain.

If there is a need to have a large amount of control over the design and features of your application, a parachain is a better choice. Keep in mind, smart contracts can be used as a testing ground before later being turned into full-fledged parachains. Smart contract platforms will usually have

convenient tooling like IDEs to facilitate quick iterations. A smart contract MVP could be created to gauge user interest before putting in the work to build out a parachain.

Each platform will have a different way of paying for and maintaining the state of your smart contract. The different patterns you may see for paying for your smart contract include:

- A transaction fee associated with deploying each transaction.
- A subscription model in which you pay some chain entity routinely for the usage of the platform.
- An access token model for which you need to hold a threshold of native tokens to use the platform (EOS has something similar). Storage rent.
- Free trial or developer promotion.
- Most smart contract platforms use some form of gas to limit the number of operations a user can perform. Users will be required to pay for the gas upfront and will be refunded for what they don't use.

You will need to consider the storage and complexity of your smart contract to ensure that gas usage stays within reasonable bounds. Storage will likely be expensive for whichever smart contract platform you use, so it is necessary to keep as much data off-chain as possible. You may consider using IPFS or Storj to keep the data and submitting only the content address on chain.

This guide now splits into two sections depending on whether you've decided on a smart contract or a parachain to build your application. Feel free to read both sections or just the one that is applicable to you.

- I want to build a parachain or parathread!
- I want to build a smart contract!

So you want to build a parachain or parathread...

Now that you have determined that building a parachain or parathread is the right approach for your new project, the next step is to decide which framework to use. Frameworks for building a parachain or parathread are known as parachain development kits (PDKs). Currently, the only PDK available is Substrate and Cumulus from Parity Technologies.

In the future, there will be many different PDKs available in different programming languages, just like there are multiple implementations of the Polkadot Host.

> **Call to Action:** Do you want to build a Parachain Development Kit from scratch? The Web3 Foundation is giving grants to teams who are doing this, learn more and apply on the W3F grants page.

Get started with Substrate

Substrate is the underlying framework on which Polkadot itself is built. It is a toolset for blockchain innovators that provides the necessary building blocks for constructing a chain. It includes a library of modular runtime plug-ins from which you can compose your chain logic.

The best way to get started with Substrate is to explore the Substrate Developer Hub, an online resource built and maintained by Parity Technologies.

How to set up your parachain

After creating your chain runtime logic with Substrate, you will be able to compile it down to a Wasm executable. This Wasm code blob will contain the entire state transition function of your chain, and is what you will need to deploy your project to Polkadot as either a parachain or parathread.

Validators on Polkadot will use the submitted Wasm code to validate the state transitions of your chain or thread, but doing this requires some additional infrastructure. A validator needs some way to stay up to date with the most recent state transitions, since Polkadot nodes will not be required to also be nodes of your chain.

This is where the collator node comes into play. A collator is a maintainer of your parachain and performs the critical action of producing new block candidates for your chain and passing them to Polkadot validators for inclusion in the Polkadot relay chain.

Substrate comes with its own networking layer built-in but unfortunately only supports solo chains (that is, chains that do not connect to the relay chain). However, there is the Cumulus extension that includes a collator node and allows for your Substrate-built logic to be compatible with Polkadot as either a parachain or parathread.

Cumulus

The goal of Cumulus is to be an extension of Substrate that will make any Substrate runtime compatible with Polkadot.

It handles the network compatibility overhead that any parachain would need to implement to be connected to Polkadot. This includes:

- Cross-chain message passing (XCMP).
- Out-of-the-box Collator node setup.
- An embedded full client of the Relay Chain.
- Polkadot block authorship compatibility.

Integrating Cumulus with your Substrate chain will port it into a parachain capable of working on Polkadot with minimal modification, possibly as little work as importing a crate and adding a few lines!

How to deploy your parachain or parathread in Polkadot

Parachain

In order to include your parachain into the Polkadot network, you will need to acquire a parachain slot.

Parachain slots will be sold in open auctions, the mechanics of which can be found on the {{ polkadot: parachain auction :polkadot }} {{ kusama: parachain auction :kusama }} page of the wiki.

Parathread

Parathreads will not require a parachain slot, so you will not need to engage in the candle auction mechanism. Instead, you will be able to register your parathread code to the relay chain for a fee and from then be able to start participating in the per-block auctions for inclusion of your state transition into the relay chain.

For more information on how parathread per-block auctions work, see the more detailed {{ polkadot: parathread :polkadot }} {{ kusama: parathread :kusama }} page.

So you want to build a smart contract...

The Polkadot relay chain itself will not support smart contracts. However, since the parachains that connect to Polkadot can support arbitrary state transitions, they can support smart contracts.

Substrate presently supports smart contracts out-of-the-box in two ways:

- The EVM pallet offered by Frontier.
- The Contracts pallet in the FRAME library for Wasm based contracts.

Frontier EVM Contracts

Frontier is the suite of tools that enables a Substrate chain to run Ethereum contacts (EVM) natively with the same API/RPC interface Ethereum exposes on Substrate. Ethereum Addresses can also be mapped directly to and from Substrate's SS58 sceme from existing accounts.

Contracts Pallet

The experience of deploying to an EVM-based chain may be more familiar to developers that have written smart contracts before. However, the Contracts pallet makes some notable improvements to the design of the EVM:

1. **Wasm**. The Contracts pallet uses WebAssembly as its compilation target. Any language that compiles to Wasm can potentially be used to write smart contracts. Although it's better to have a dedicated domain-specific language and for that reason Parity offers the ink! language.

2. **Rent**. Contracts must pay rent or else hold a deposit suitably large enough in order to justify its existence on-chain. When a contract does not uphold this, it may create what's called a *tombstone* which is a reference to the contract. In some conditions, the contract will be deleted outright along with its storage if it does not maintain these requirements.

3. **Caching**. Contracts are cached by default and therefore means they only need to be deployed once and afterward be instantiated as many times as you want. This helps to keep the storage load on the chain down to the minimum. On top of this, when a contract is no longer being used and the *existential deposit* is drained, the code will be erased from storage (known as reaping).

Ink

ink! is a domain specific language for writing smart contracts in Rust and compiles to Wasm code. As it states in its README, it is still in an experimental phase so brave developers should be aware that they might have a bumpy - but workable - development experience. There are some projects that have built projects in ink! with a decent level of complexity such as Plasm's Plasma contracts, so it is mature enough to start building interesting things.

For interested developers, they can get started writing smart contracts using ink! by studying the examples that were already written. These can be used as guideposts to writing more complex logic that will be deployable on smart contract parachains.

ink! has laid much of the groundwork for a new smart contract stack that is based on a Wasm virtual machine and compatible with Substrate chains.

It's still early

It's still very early for smart contracts on Polkadot and the development is only now stabilizing. We are actively producing content to help developers get up to speed and will maintain the wiki with the latest resources. You should also keep up to date with the following links:

- Edgeware.
- Moonbeam
- ink!. (Keep an eye out for content on the wiki tab.)
- Substrate contracts pallet.

Edgeware

One project that is live today with the smart contracts pallet is Edgeware. Edgeware is a permissionless platform for smart contracts and is conducting experiments with on-chain governance. It is currently the best option for developers who have created their smart contracts and want to deploy to a live environment.

Edgeware intends to at some point connect to Polkadot as a parachain that allows for smart contracts. At this point, the smart contracts would be able to interact with other pieces of the Polkadot ecosystem through XCMP.

Edgeware general documentation can be found here and how to deploy smart contracts on Edgeware here.

Moonbeam

Moonbeam is another project that is planning to deploy to Polkadot as a parachain and will support smart contracts. Since Moonbeam uses Frontier, an interoperability layer with existing Ethereum tooling, it will support all applications that are written to target the EVM environment with little friction.

Try deploying a contract to Moonbeam by following their documentation.

Conclusion

This guide has given you a mental model and shown the requisite resources to help you determine and start building your project as a parachain or smart contract today. Even though the tooling is still maturing, the advantage of being early will be the familiarity and head start on your project, allowing you to innovate and create something truly new.

If you have interesting ideas for parachains or smart contracts on Polkadot feel free to drop into the Polkadot Watercooler to talk about them. Developers may be interested in joining the Polkadot

Beginners Lounge or Substrate Technical to ask their questions. As always, keep up to date with Polkadot and Kusama by following the social channels.

Good luck!

Rococo Parachain Testnet

Rococo is a Polkadot testnet built for testing parachains. Rococo utilizes Cumulus and HRMP (Horizontal Relay-routed Message Passing) in order to send transfers and messages between parachains and the Relay Chain. Every message is sent to the Relay Chain, then from the Relay Chain to the desired parachain. Rococo currently runs four test system parachains (Statemint, Tick, Trick, and Track), as well as several externally developed parachains.

What Parachains are on Rococo Now?

You can see the list of included parachains here. A list of proposed parachains is available here.

Obtaining ROC

ROC are available in the Rococo Faucet channel on Matrix. To receive ROC tokens, use the command:

```
!drip YOUR_ROCOCO_ADDRESS
```

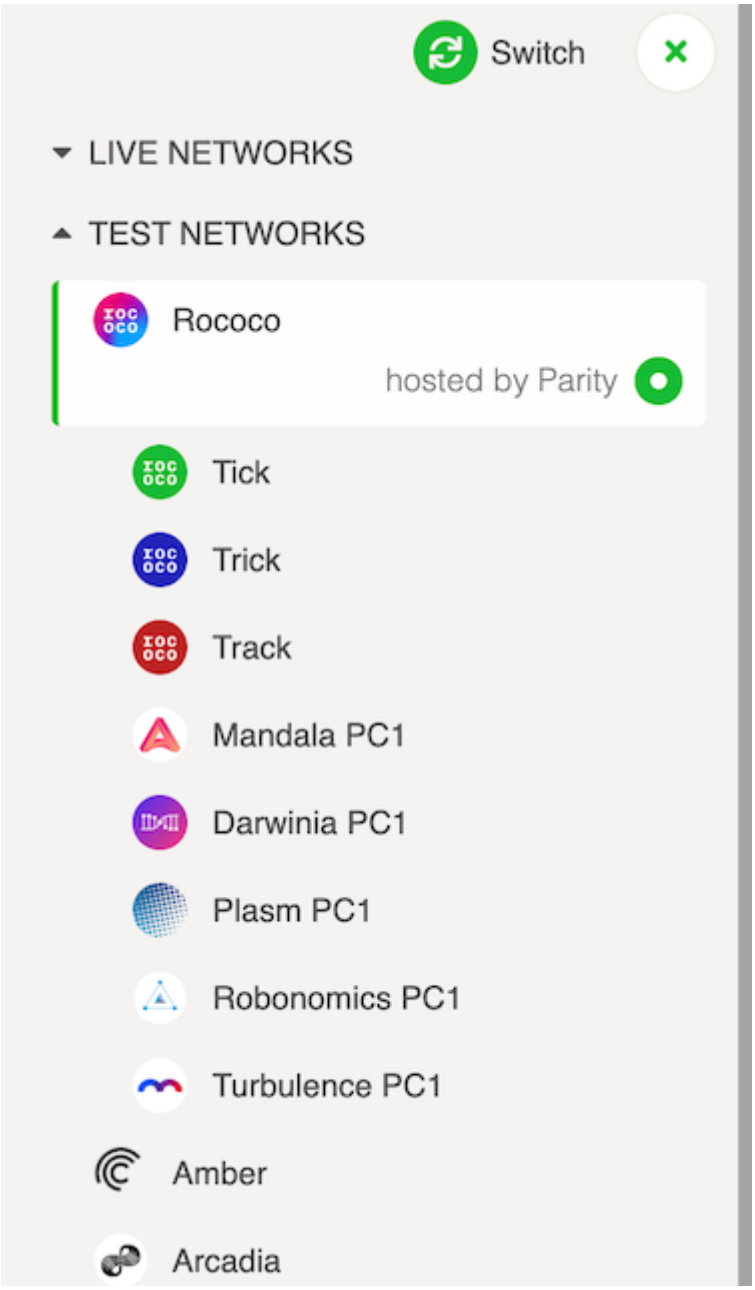
Build and Register a Rococo Parathread

Cumulus is set of tools for writing Substrate-based parachains.

If you are interested in running and launching your own parathread or parachain, Parity Technologies has created a cumulus parachain workshop to show you how. Get stuck or need support along the way? Join the Rococo matrix chat channel and connect with other builders there.

How to connect to a Parachain

If you would like to connect to a parachain via Polkadot-JS Apps, you may do so by clicking on the network selection at the top left hand corner of the navigation and selecting any parachain of choice. For the purpose of these following examples, we will be using the Rococo testnet "Custom Node" underneath "Development", following the parachain workshop.



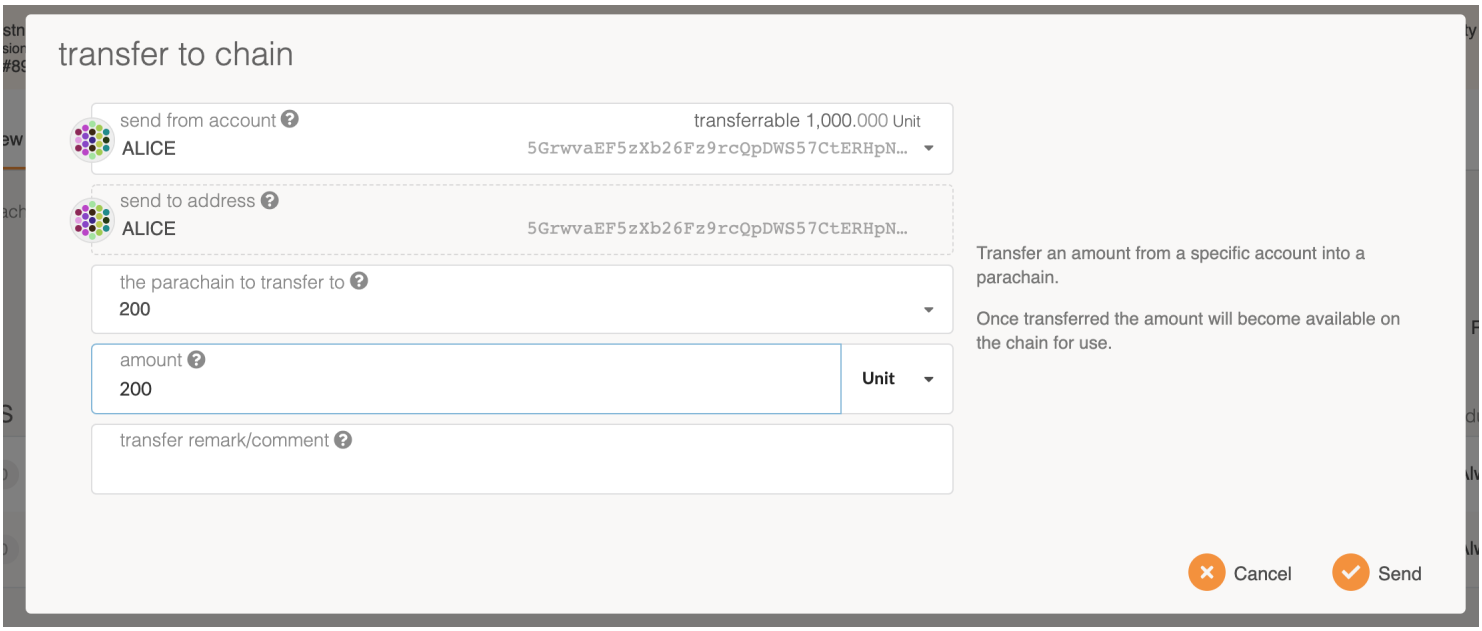
How to make Cross Chain transfers

To send a transfer between parachains, navigate to "Accounts" > "Transfer". From here, you'll need to select the parachain node that you are running. Next, enter in the amount that you'd like to send to another parachain. Be sure to select the correct parachain you'd like to send an amount to. Once you've hit the "Submit" button, you should see a green notification, indicating a successful transfer.

Downward Transfers

Downward transfers are when an account on the Relay Chain sends a transfer to their account on a different parachain. This type of transfer uses a depository and mint model, meaning that when the DOT leave the sender's account on the Relay Chain and are transferred into an account on a parachain, the parachain mints a corresponding amount of tokens on the parachain.

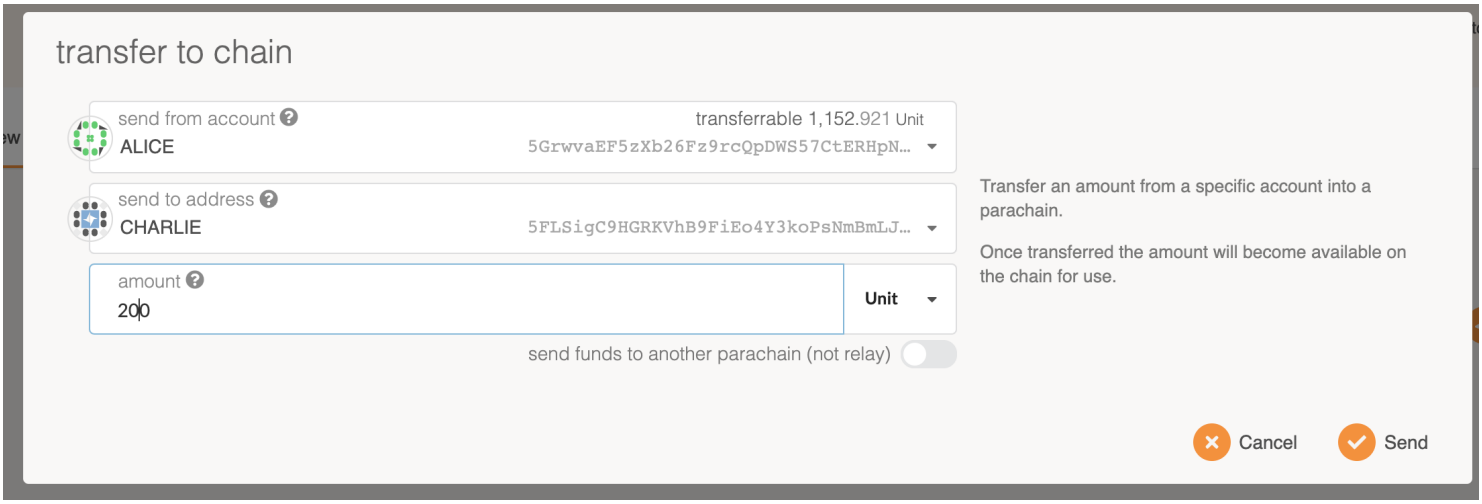
For example, we can send tokens from Alice's account on the Relay Chain to her account on parachain 200. To do so, we will need to head to the "Network" > "Parachains" tab and click on the "Transfer to chain" button.



Notice here, that we can select which parachain to send the funds to, specify the amount to be sent, and add any comments or a memo for the transfer.

Upward Transfers

Upward transfers occur *from* a parachain *to* an account on the Relay Chain. To proceed with this kind of transfer, we need to be connected to a parachain node on the network and be on the "Network" > "Parachains" tab. Click on the "Transfer to chain" button.



Note that the toggle should be set to off, ensuring that the funds go to the Relay Chain and not another parachain.

Lateral Transfers

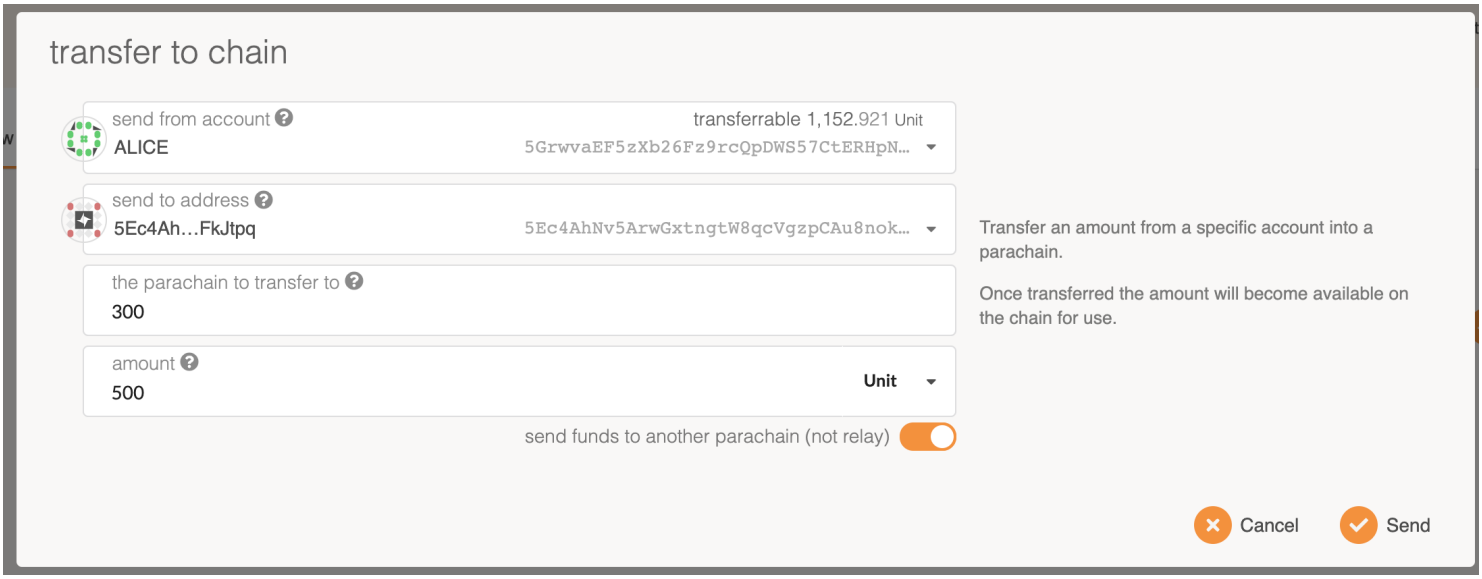
This type of transfer is only possible with at least two different registered parachains. In true XCMP, lateral transfers would allow for messages to be sent directly from one parachain to another. However, this is not yet implemented, so the Relay Chain is helping us deliver messages for the time being. Lateral transfers work through the depository model, which means that in order to transfer tokens from chain 200 to chain 300, there must already be tokens owned by chain 200 deposited on chain 300. Lateral transfers are called HRMP, Horizontal Relay-Chain Message Passing.

Before we can actually send funds from one parachain to another, we must ensure that the chain's account on the recipient chain has some funds in it. In this example, Alice will be sending some funds from her account on parachain 200 to her account on parachain 300.

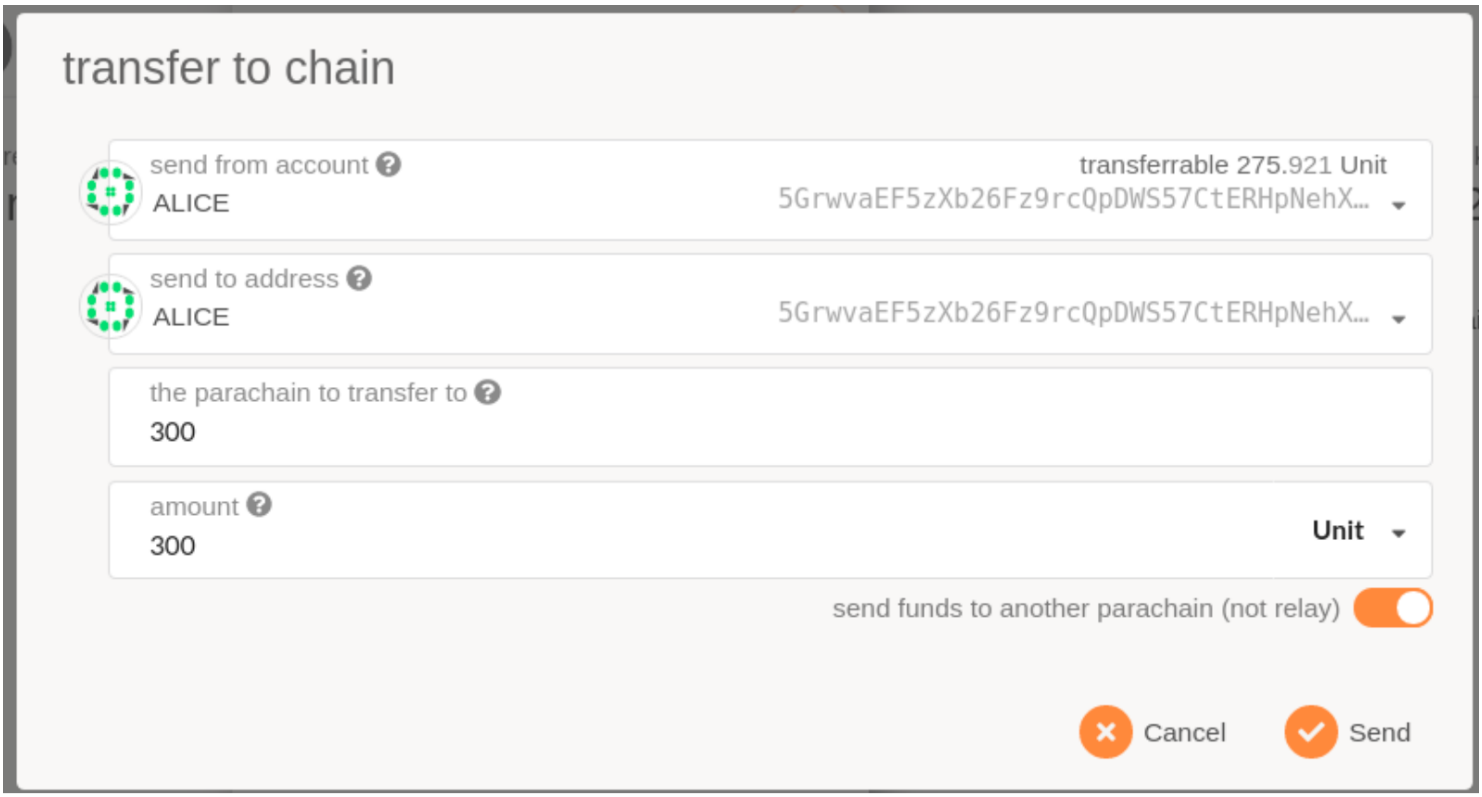
We can get that parachain account address, from our parachain 300's terminal:

```
2020-08-26 14:46:34 Parachain Account:
5Ec4AhNv5ArwGxtngtW8qcVgzpCAu8nokvnh6vhtvvFkJtpq
```

From Alice's account on the Relay Chain, she is able to send some amount to parachain 200's depository.

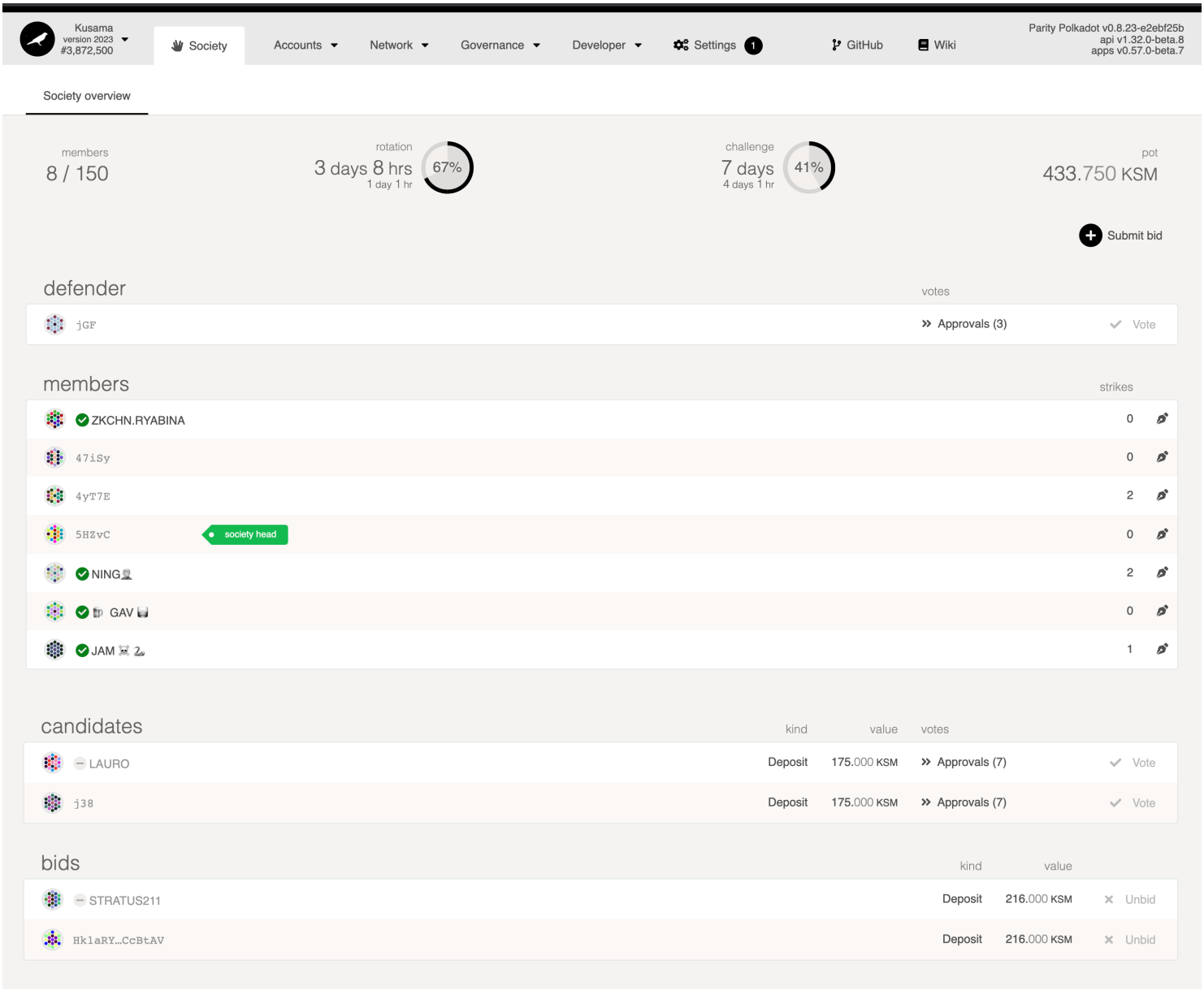


Alice is now able to send from her account on parachain 200 to her account on parachain 300.


transfer to chain
send from account
ALICE
transferrable 275.921 Unit
5GrwvaEF5zXb26Fz9rcQpDWS57CtERHpNehX…
send to address
ALICE
5GrwvaEF5zXb26Fz9rcQpDWS57CtERHpNehX…
the parachain to transfer to
300
amount
300
Unit
send funds to another parachain (not relay)
Cancel
Send

Join Kappa Sigma Mu

Kappa Sigma Mu is a membership club using the Substrate Society pallet. It is an economic game to incentivize users to join a society that coordinates around whatever the rules are decided to be. The members of the society are incentivized to participate in the society via the rewards paid by the treasury. Currently, there is only one society on Kusama but it is possible to have multiple societies in the future through a runtime upgrade.



Before joining the society, let's take a brief look at the Society UI on Polkadot-JS apps and read through all the rules to become a member.

UI Overview

- `Members`: The number of members in the society. Currently, the maximum number of members is set to `150`. It can be changed by using governance to increase the number.
- `Rotation`: The time period for membership rotations.
- `Challenge`: The time period to randomly select one of the members to defend his membership in the society.
- `Pot`: Resource balance that is used to support members of the society.
- `Bids`: A list of users who submitted a bid to join the society.

User Types

Below are the various types of users at different stages.

- `Bidder` - A token holder who intends to join the society by placing a bid.
- `Candidate` - The selected bidders that will be voted on by members of the society.
- `Suspended Candidate` - The candidates that failed to join the society.
- `Member` - Member of the society.
- `Suspender Member` - A member of the society who has accumulated too many strikes or failed their membership challenge.
- `Head` - One winning candidate will be randomly chosen as head of the members, weighted by the number of approvals the winning candidates accumulated.
- `Defender` - In every challenge period, one of the members will be randomly selected to defend their membership in the society. The rules for defending the membership are documented in the rules.

Procedure

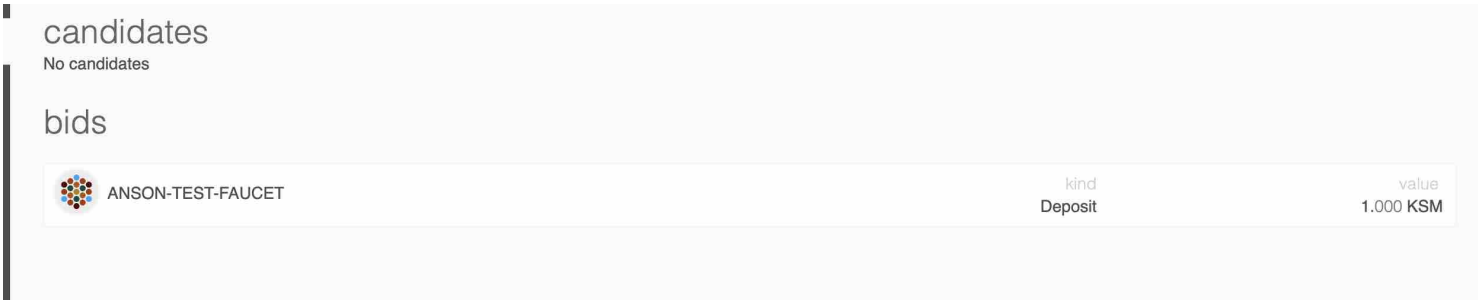
Remember to take a look at the rules first. And since those rules are not enforced entirely on-chain, it is recommended to join the Kappa Sigma Mu Lounge to ask any questions if anything is unclear.

1. Bid Phase

To submit a bid, click the Submit Bid button on the Society page.

Anyone can submit a bid to join the society by reserving a deposit or finding an existing member to create a bid on their behalf by vouching for them. At every rotation period, as many bids as the society pot can support will be selected. The selected bids will be moved to the candidate phase, whereas bids that were not selected will stay in the bidder pool until they are selected or a user chooses to unbid.

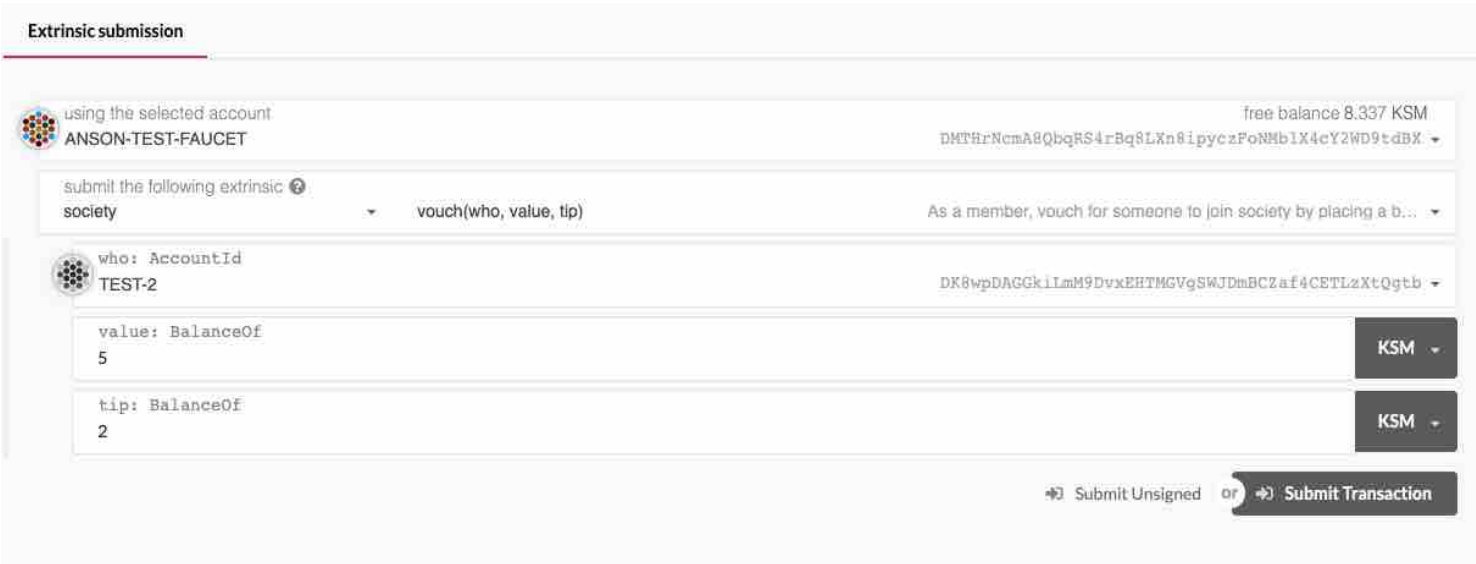
Anyone who wants to join the society is required to deposit 1.6 KSM for reserve on Kusama and declare the bid amount (1 KSM in this case) that they will receive for joining the society.



Once you have submitted the transaction, your bid will be shown on the Society page under the bids section. You can cancel the bidding if you changed your mind about joining the society by calling `unbid` on the same page.

You can find an existing member to place a bid on your behalf if you do not have KSM and you are willing to give them a tip. An existing member can submit a `vouch` transaction through the Extrinsics page.

> vouch(who,value,tip)
>
> who - The user you are vouching for
>
> value - The value that the user would like to get when joining the society
>
> tip - Fees you get
>
> Note: The final value that the candidate will get = (value - tip)



2. Candidate Phase

Bids selected in this phase will be voted on by the existing members to decide whether or not you will be approved to join the society. Members will vote for all the candidates and the final outcome will be randomly selected by one of the votes. Let's take a look the example shown below:

> Note: If the randomly selected member does not vote, it will be treated as a rejection. For each rotation period, the maximum number of members that can be accepted is set as 10.

A - Accept, R - Reject, S - Skeptic

Member	1	2	3	4	5
Vote	A	A	A	R	S
Selected			X		

In this example, a candidate will be approved to join the society since member 3 was selected as a final voting outcome. A number of members will also be randomly chosen as "skeptics" to vote for the candidates during the rotation period.

Since member 5 was chosen as a skeptic, they are required to participate in the voting process. If they do not participate in voting, they will be punished with one strike per missing vote. If one accumulates too many strikes, one's membership is suspended which means they may need to re-apply and their unclaimed payouts will be slashed. Moreover, each member who voted opposite to the randomly selected vote will have their unclaimed payouts slashed and strikes increased. In this case, member 4 will be punished.

> Note: The maximum number of strikes you can have is on Kusama is 10.

The slashed funds (2 KSM currently) will be given to a random member who voted the same as the selected vote as a reward for participating in the vote. The reward is escrowed for some period of time - see below.

Lock-up Time

It would take the number of members of the society as the variable to determine how many blocks you have to wait in order to get the payout. The longest lock-up time is close to 3 years. The formula is defined in the society pallet if you would like to have a look.

Example:

```
Let's assume we have 5 members in the society.

lock_duration = 100 - 50_000 / (5 + 500)
lock_duration * MAX_LOCK_DURATION_IN_BLOCKS

Result = 1% * 15_552_000 ~ 11 days
```

Based on the above calculation, it is required to wait close to 11 days to get the slashed funds.

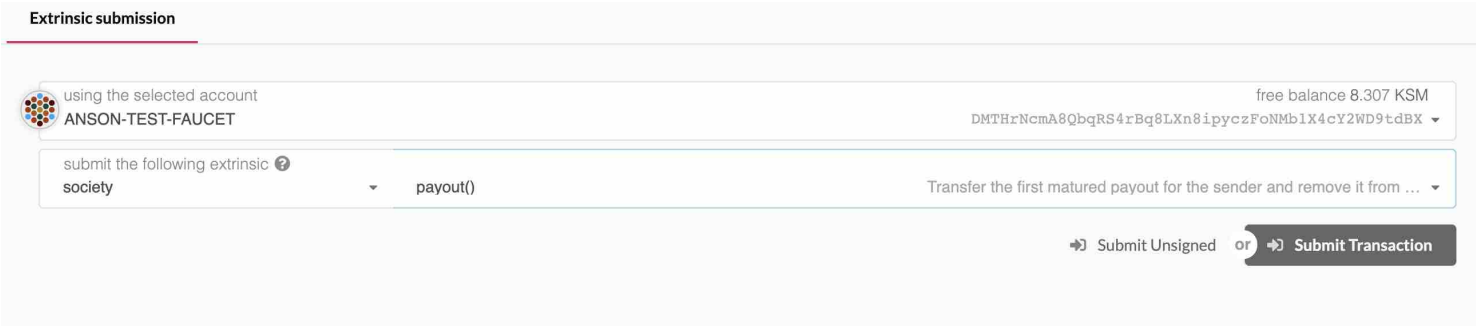
If the candidate wins the vote, they receive their bid reward as a future payout. If the bid was placed by a voucher, they will get back the reward that was set during vouching with the remainder given to the candidate - both escrowed for some time.

If the candidate loses the vote, they are suspended and it is up to the founder of the society (the `Suspension Judgement Origin`) to determine if the candidate should go through the bidding process again, should be accepted into the membership society, or rejected and their deposit slashed.

3. Member Phase

Once you become a member of the society, you will get back the deposit that you have reserved during the bidding. A few things you need to be aware of. First, you should vote on candidates who applied for the membership in every rotation period.

Second, you will need to claim your payout manually by calling `payout` after the lock-up time. It is the same as the above mentioned lock-up formula.



Third, there will be a membership challenge every seven days on Kusama. So one of the members will be randomly selected as a defender. Then, other members can vote whether this defender should stay in the society or not. A simple majority wins the vote. You can take a look here and search for "Existing Members (Challenges)". Besides that, you can earn extra KSM by helping a user apply for the membership and requesting a tip. This is useful when a user does not have enough balance to reserve a deposit. The tip will be given when a user successfully joins the society.

> Note: Each member can only vouch for one user at a time. A member is not required to reserve the deposit when vouching for a user.

If a member accumulates too many strikes or fails their membership challenge, they will become suspended. While a member is suspended, they are unable to claim matured payouts. It is up to the suspension judgment origin to determine if the member should re-enter society or be removed from society with all their future payouts slashed.

Useful links

Convention of Approval of Membership - Rules about joining the Kusama society

Social Recovery

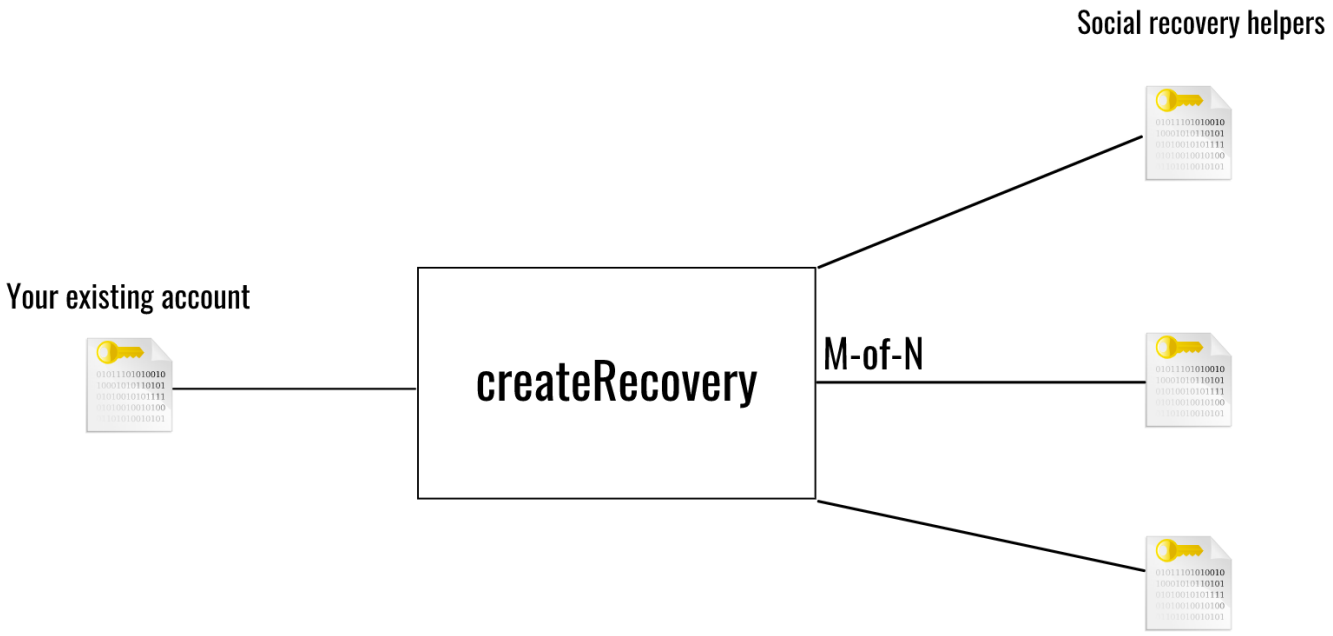
Managing an account is not an easy task. Many people have lost their private key due to improper key management over the past few years. Kusama provides a method that allows users to recover their accounts by setting up a social recovery. It is an M-of-N recovery tool that is based on the multisignature wallet to get back access of your lost account.

> Note: There is no way to get back your private key by using this method. This is just a way of performing transactions on behalf of the lost account, so you can think of it like a proxy instead.

In this guide, you will learn how to create a recoverable account, how to recover it, and what you need to be aware of when using it.

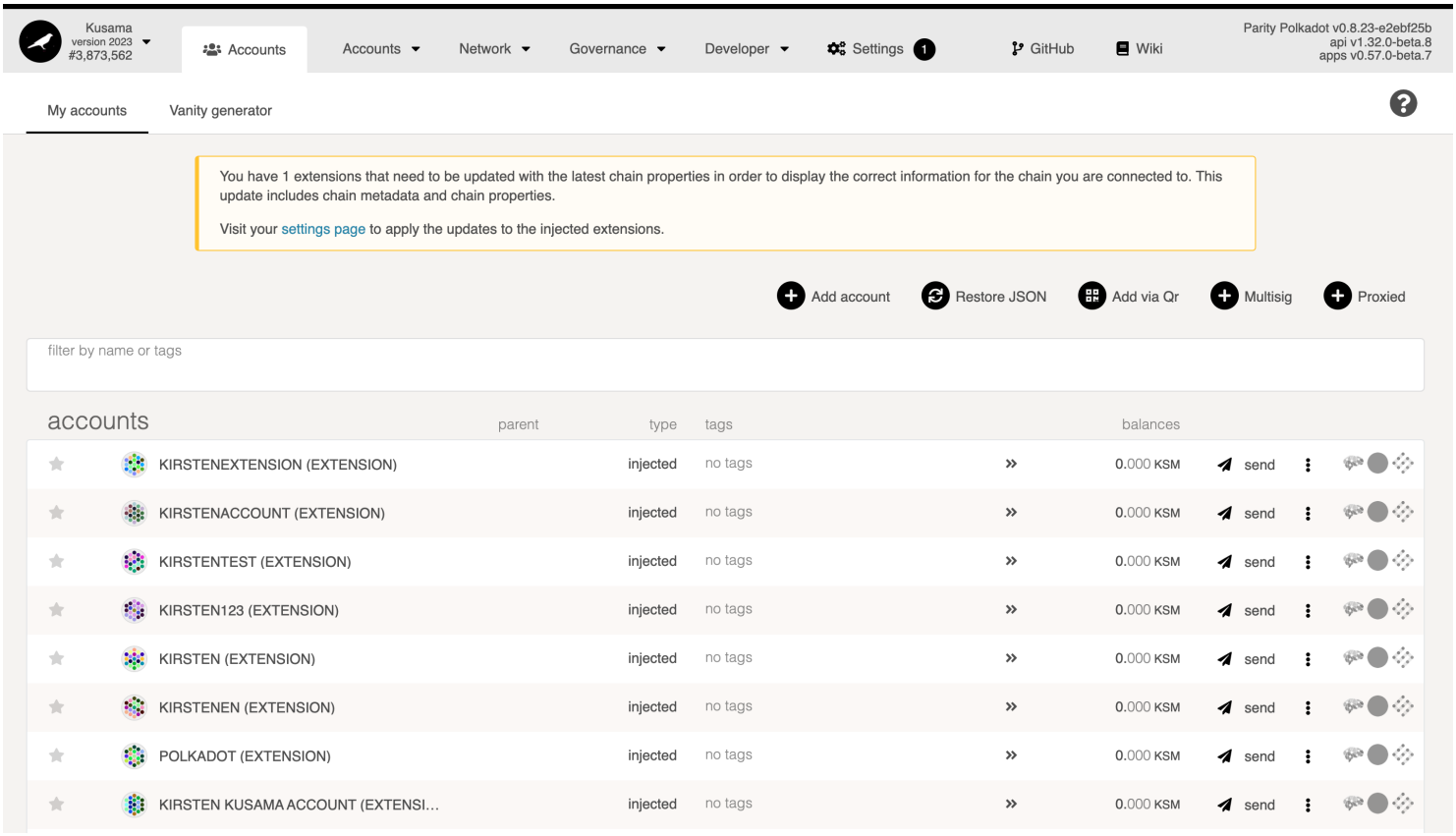
Create a Recoverable Account

You will use your existing account to call `createRecovery` to select a list of accounts that you trust to help you recover the account when you lose the private key. To create a recoverable account, you will be required to set a `threshold` that is the number of your friends who need to approve the recovery process in order to recover your account.

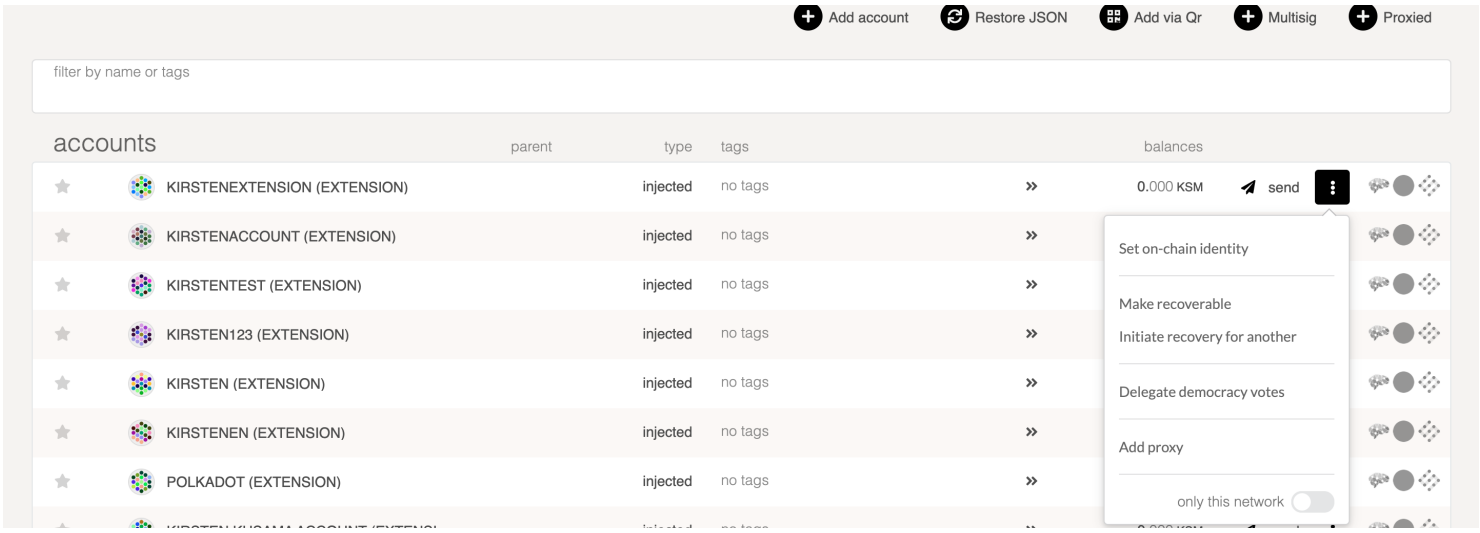


> Note: If you are recovering an account, ensure that your network is set to Kusama. You can do this by selecting the network from the top right corner in the Polkadot-JS UI.

First, go to Polkadot-JS -> Accounts -> Accounts page that shows all available accounts on your browser's local storage and Polkadot-JS extension. To create a recoverable account, make sure that you have some KSMs to pay for the transaction fees. You will also need some for the reserve required by the account recovery setup.



Then click the menu that is located besides the "send" button, and choose "Make recoverable".



Now you need to provide the following information:

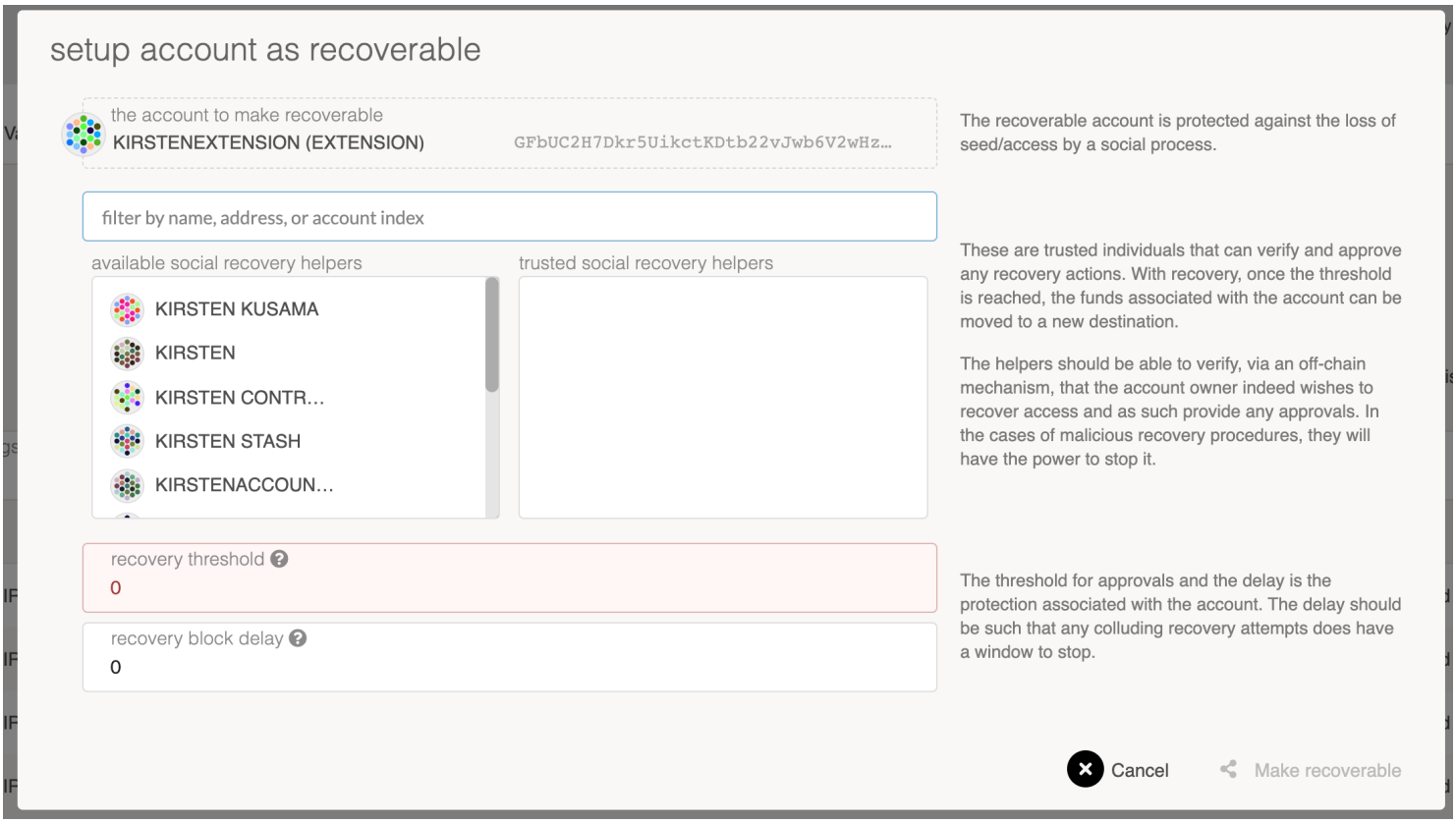
`trusted social recovery helpers` - A list of accounts that you trust. These can help you if you lose the private key. Since setting up a recoverable account requires you to lock up KSM, ensure your account has enough transferable balance to cover it. As you select additional recovery helpers, more KSM will be required.

`recovery threshold` - The number of friends required to submit a `vouchRecovery` transaction in order to recover the account.

> Note: 1 is the minimum, but it is not recommended to set a small number. If you set 1, that means any of your recovery helpers would be able to recover your account.

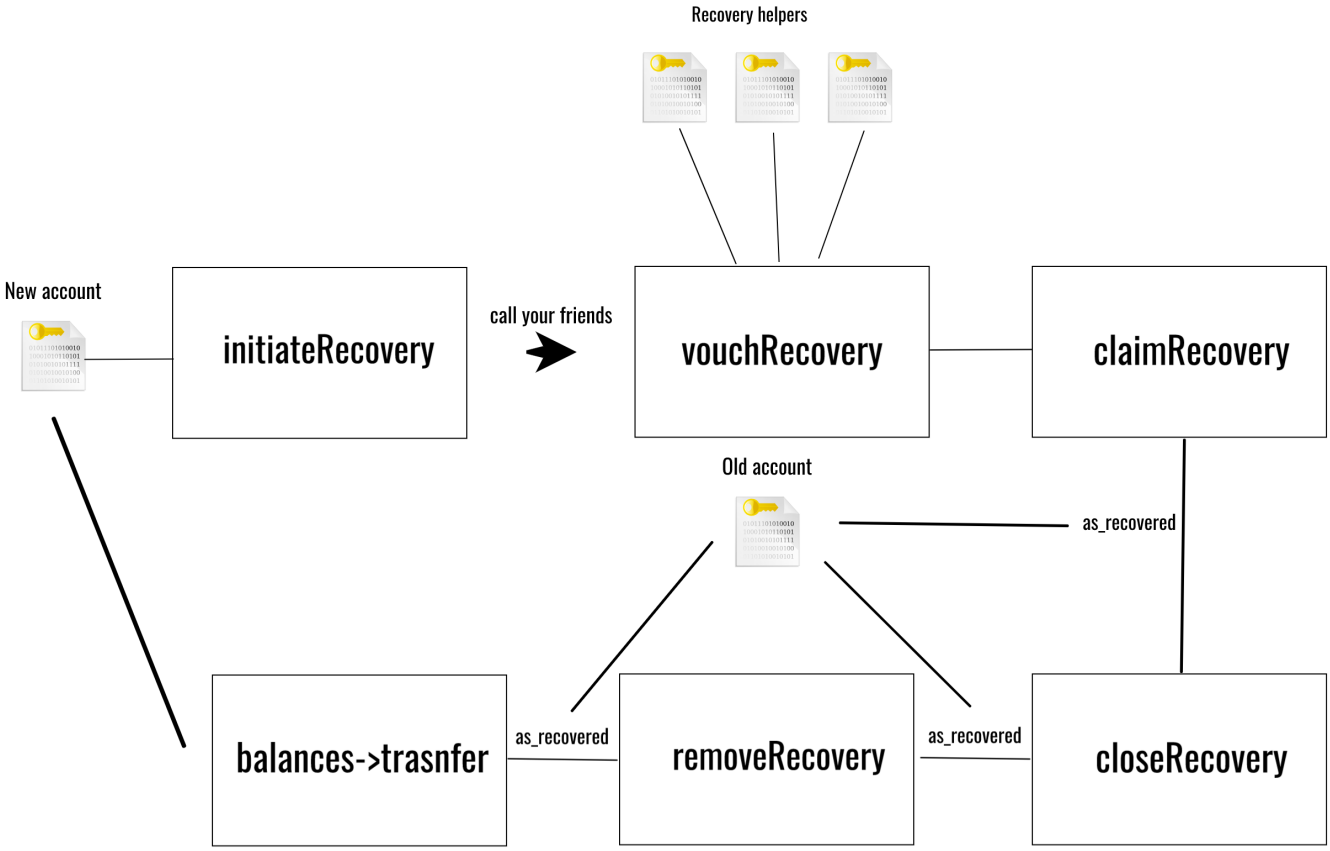
`recovery block delay` - Once the threshold is reached, you will need to wait until the block delay has passed until you can claim the recovery. This is a protection mechanism to allow the account owner to have enough time to check and react in case someone pretends to be you and initiates a recovery process.

> Note: Setting the block delay to be a little longer would be better since even if an attacker acquired enough signatures to recover your account, they would still have to wait until the block delay is passed in order to control your account.



Recover your Account

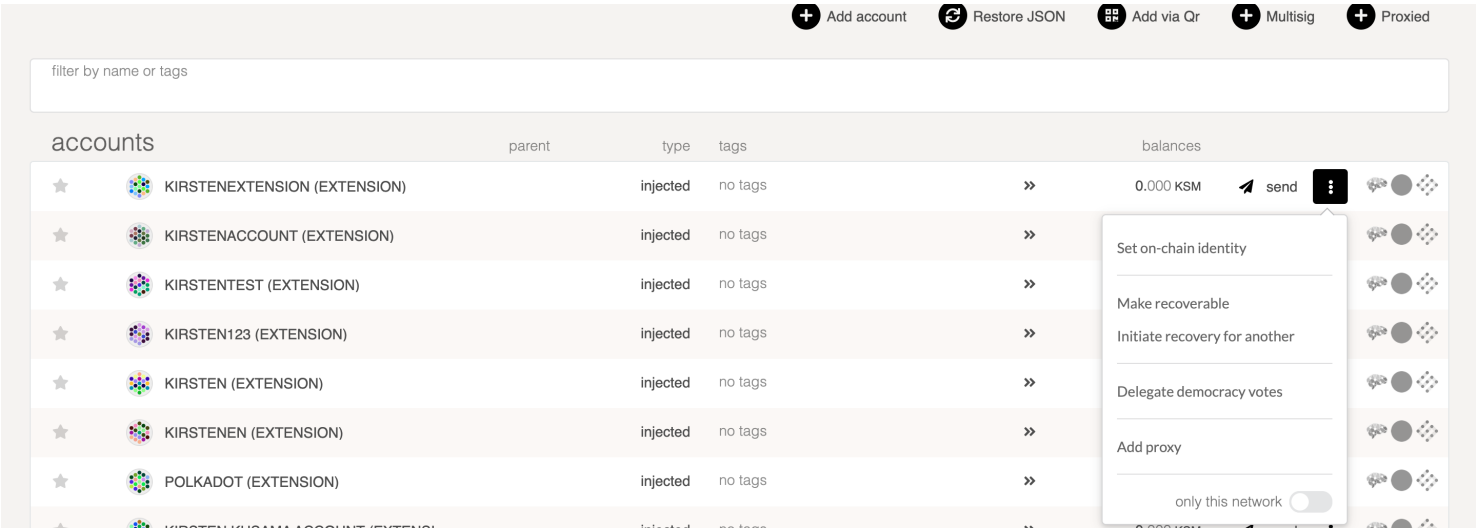
This section would be showing you how to initiate a recovery process and get back the balance that held in the lost account to the new account.



The above diagram shows the whole process of recovering an account.

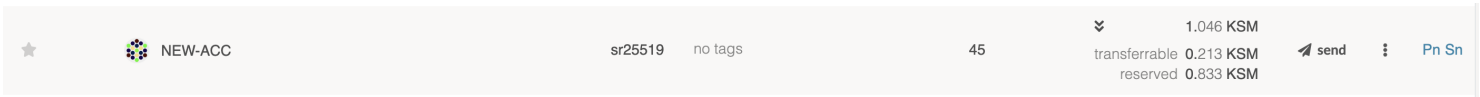
> Note: Ensure that your new account has enough KSM to pay for the transaction fees and the amount that is used for reserve when doing the recovery.

Navigate to the menu that is located besides the send button in the row of your new account and click the "Initiate recover for another" option.

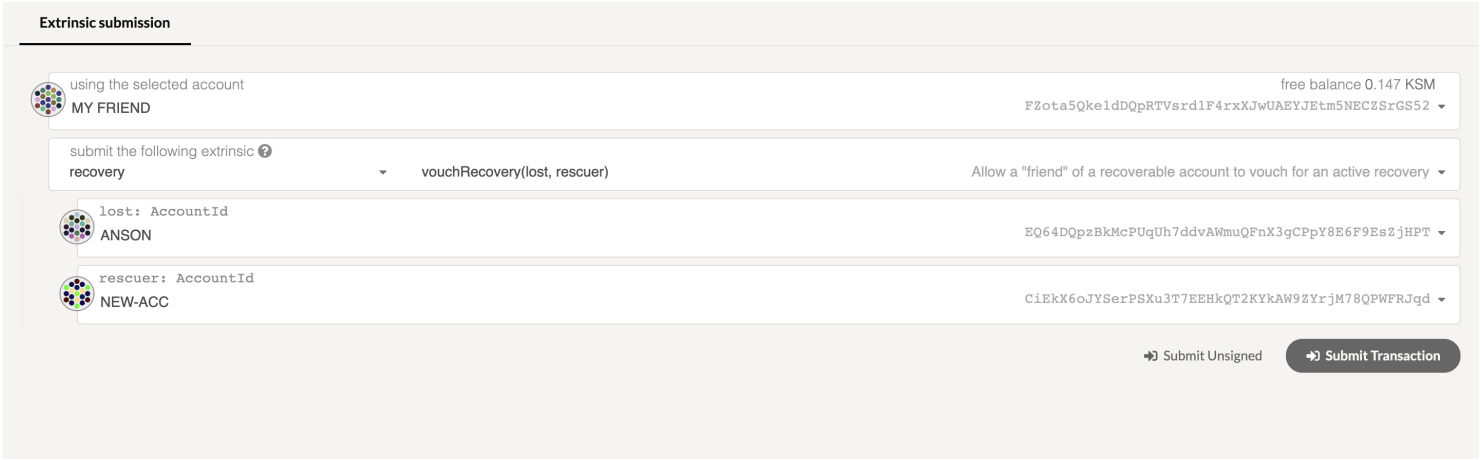


Then input the address you would like to recover in "recover this account" field and click "Start recovery".

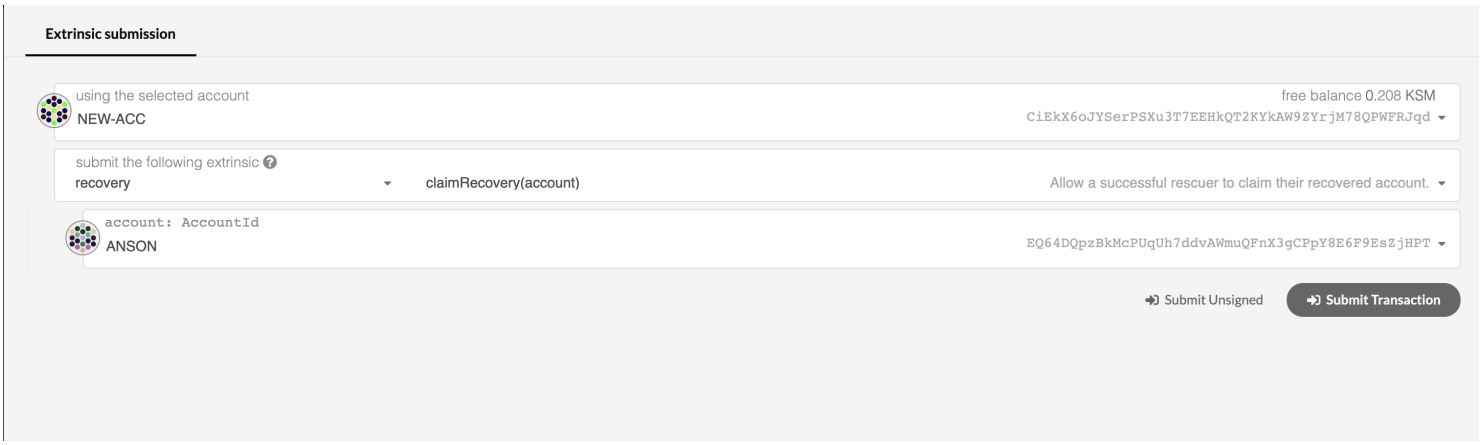
Once the transaction went through, some KSM will be locked to prevent malicious behavior.



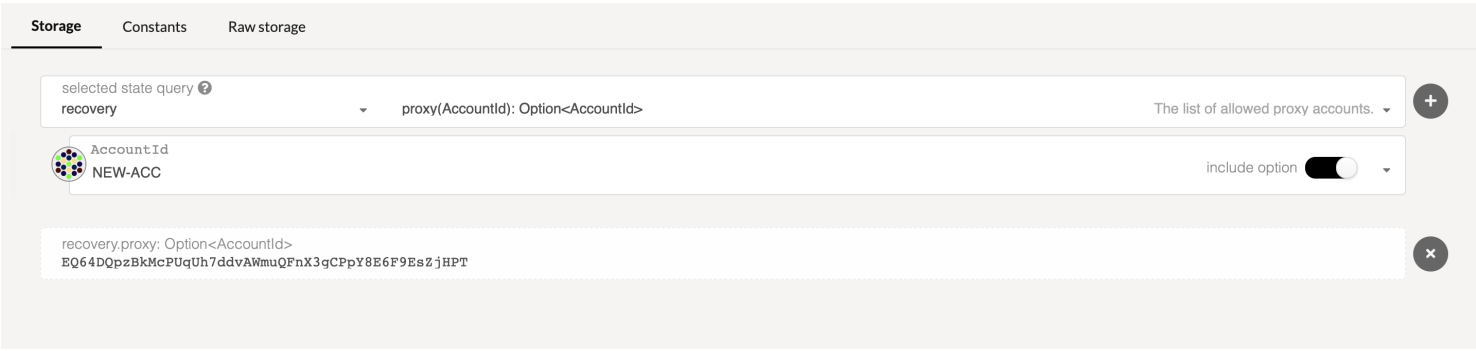
Now call your friends that you have set in the first section, but heading over to "Developer" > "Extrinsics" and using the recovery pallet. They are required to submit a `vouchRecovery` transaction.



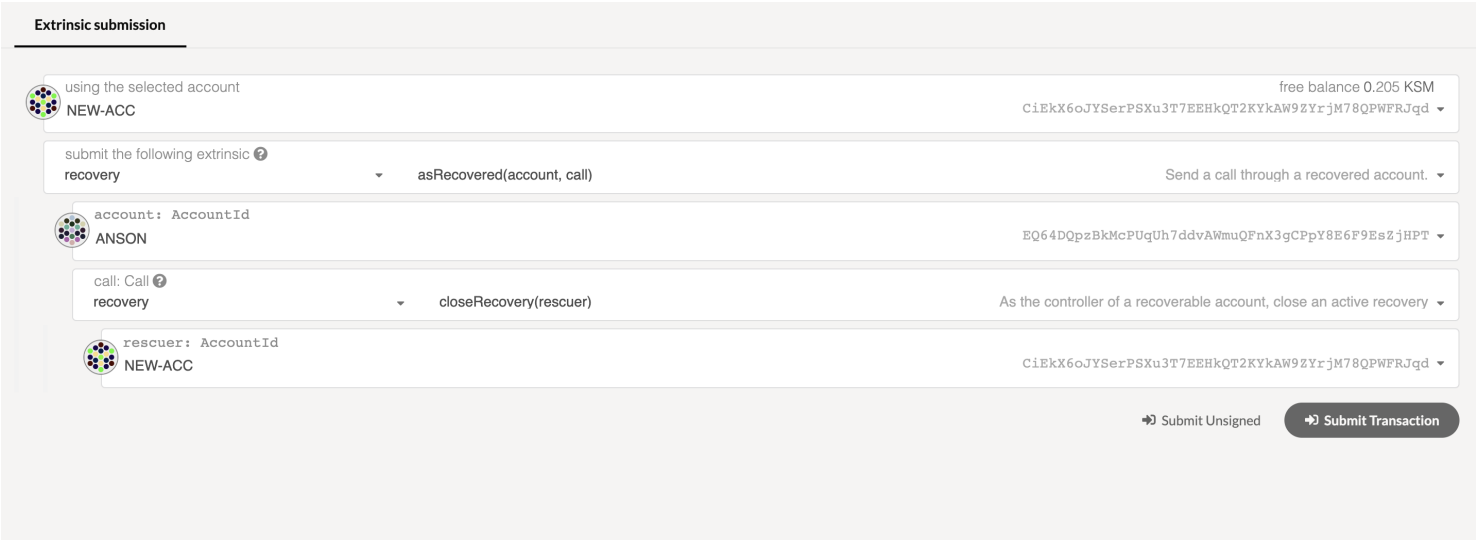
Once the threshold is reached and the block delay is passed, use the new account to submit a `claimRecovery` transaction that would set a proxy on behalf of your lost account. It means that you can still use the lost account in an indirect way to interact with the network.



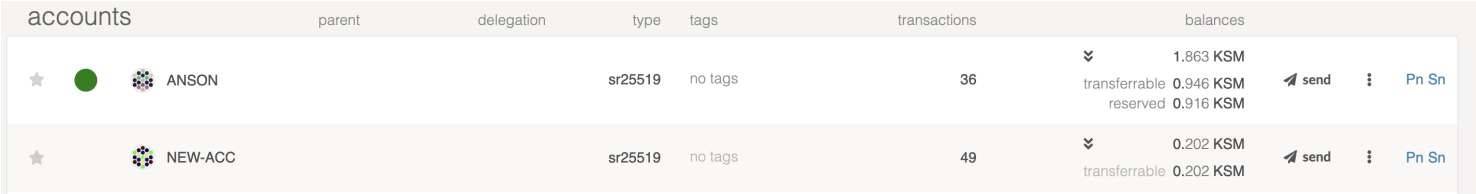
To see the proxy information, use your new account by calling the "recovery->proxy(Accountid)" function at the Chain state page. It should point to your lost account.



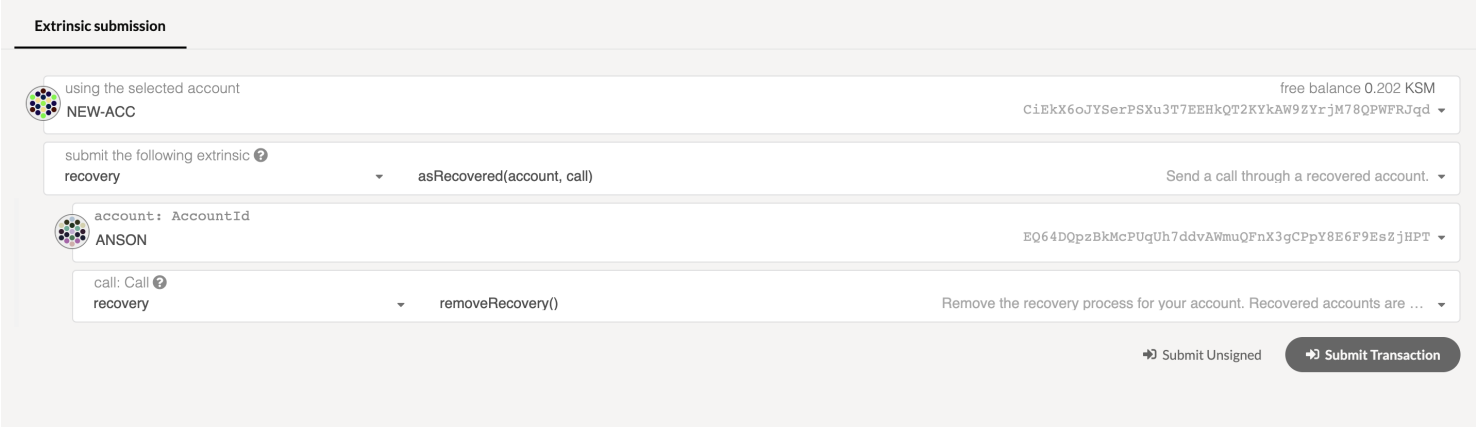
Next, in order to call the "closeRecovery" transaction, you can make use of the "asRecovered" function as your lost account to get the locked KSM.



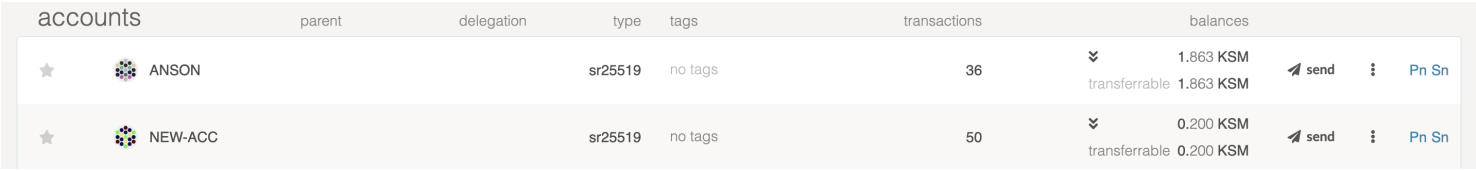
Once the transaction goes through, the reserved KSM from the "NEW-ACC" will have been moved to the lost account. This is a way of preventing someone from recovering other accounts maliciously. Imagine if someone tried to initiate recovery on your account, you can do this to slash their locked KSM.



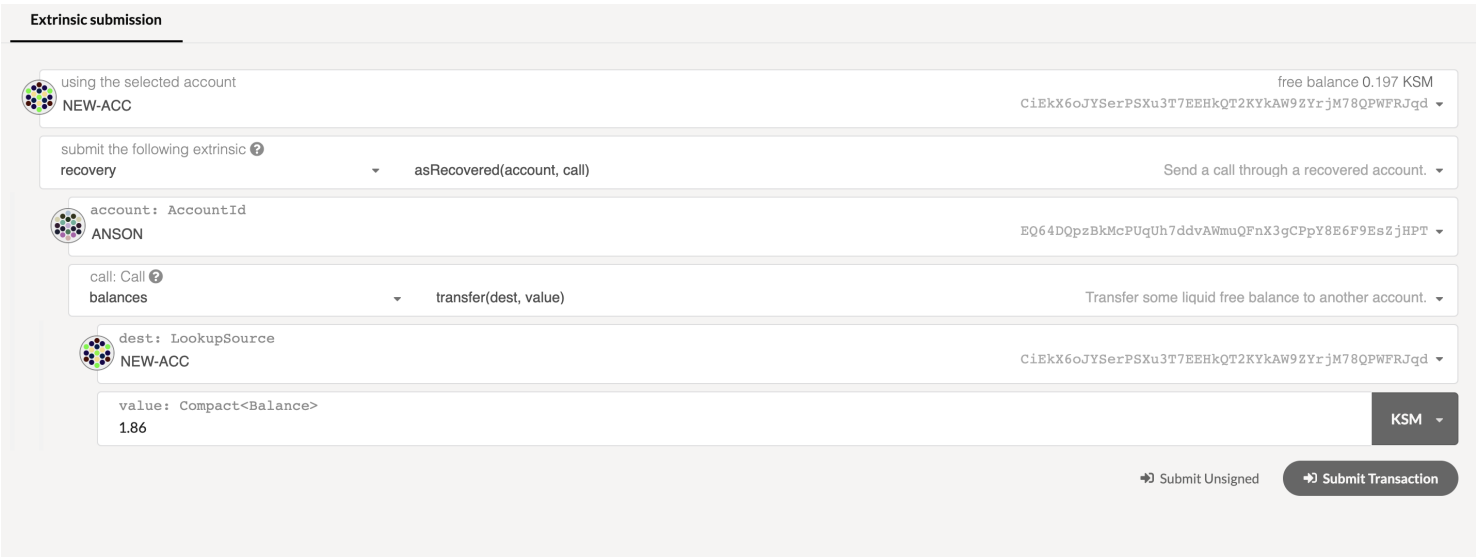
Moving on, we use the `asRecovered` function to submit the `removeRecovery` transaction on behalf of the lost account to release the reserved KSM from your lost account.



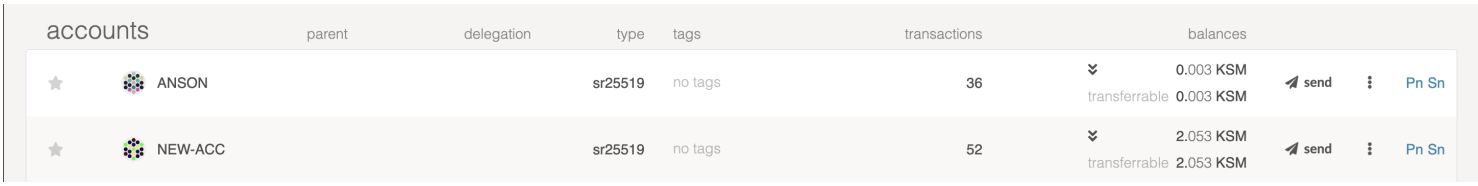
Now all your account balance should be transferable.



Finally, transfer all of your available balance from the lost account to the new account.



The recovery process is now complete and successful.



> Note: There is still one possible way to recover the account without going through the recovery process. That is by using the `Root` origin. However, in order to use root permissions you will need to either go through the council or submit a public proposal. To learn more about governance, see here.

Further Reading

- Substrate's Recovery Pallet - The Rust implementation of the recovery pallet.

Errors and How to Resolve Them

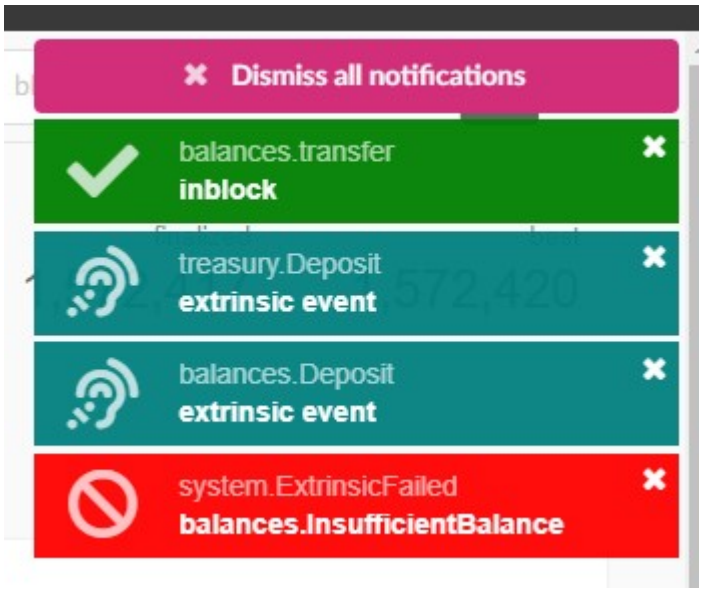
Errors in Substrate-based chains are usually accompanied by descriptive messages. However, to read these messages, a tool parsing the blockchain data needs to request *chain metadata* from a node. That metadata explains how to read the messages. One such tool with a built-in parser for chain metadata is the Polkadot-JS Apps UI.

If this page does not answer your question, try searching for your problem at the Polkadot Knowledge Base for more information on troubleshooting your issue.

PolkadotJS Apps Explorer

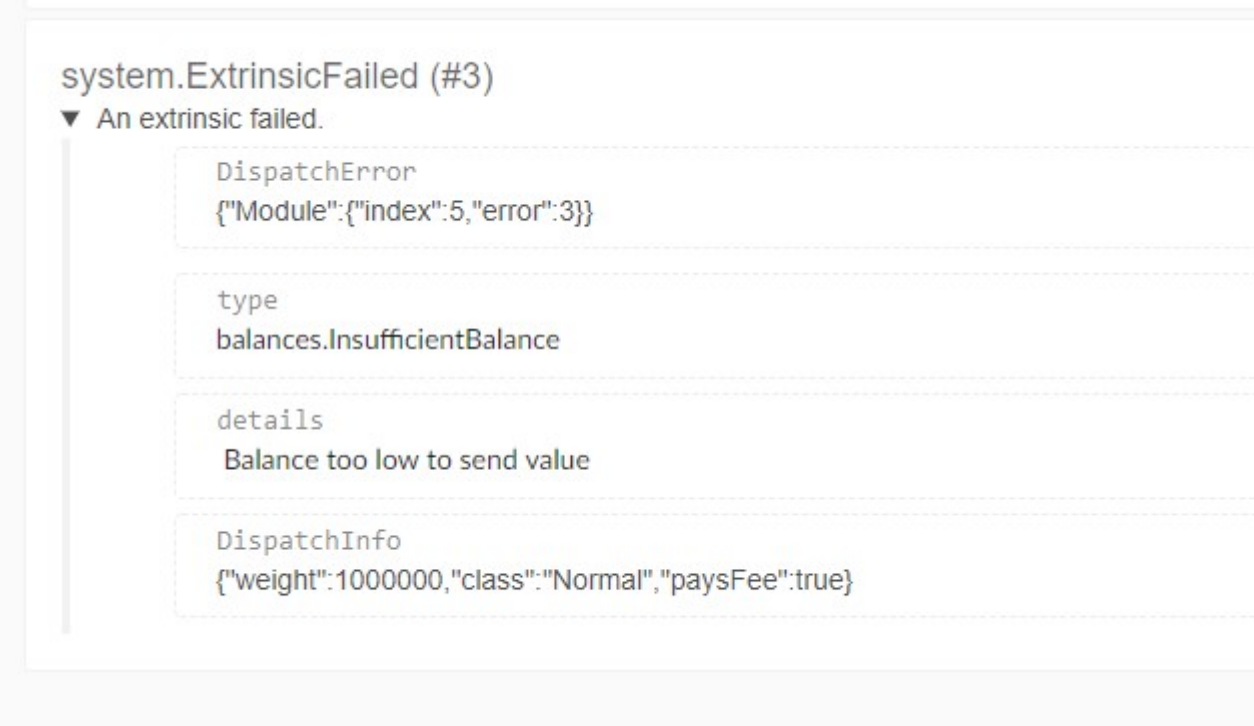
Here's how to find out the detailed error description through Polkadot-JS Apps.

A typical failed transactions looks something like this:



The image displays only the error name as defined in the code, not its error message. Despite this error being rather self-explanatory, let's find its details.

In the explorer tab, find the block in which this failure occured. Then, expand the `system.ExtrinsicFailed` frame:



Notice how the `details` field contains a human-readable description of the error. Most errors will have this, if looked up this way.

This block is a live example of the above.

If you cannot look up the error this way, or there is no message in the `details` field, consult the table below.

Polkascan and Subscan

Polkascan and Subscan show the `ExtrinsicFailed` event when a transaction does not succeed (example). This event gives us the `error` and `index` indices of the error but does not give us a nice message to understand what it means. We will look up the error in the codebase ourselves to understand what went wrong.

First, we should understand that the `index` number is the index of the pallet in the runtime from which the error originated. The `error` is likewise the index of that pallet's errors which is the exact one we're looking for. Both of these indices start counting from 0.

For example, if `index` is 5 and `error` is 3, as in the example linked above, we need to look at the runtime for the fourth error (index 3) in the sixth pallet (index 5).

By looking at the runtime code we see that the pallet at index 5 is "Balances". Now we will check the Balances pallet's code which is hosted in the Substrate repository, and look for the fourth error in the Error enum. According to its source the error that we got is `InsufficientBalance` or in other words "Balance too low to send value".

Common Errors

The table below lists the most commonly encountered errors and ways to resolve them.

Error	Description	Solution
BadOrigin	You are not allowed to do this operation, e.g. trying to create a council motion with a non-council account.	Either switch to an account that has the necessary permissions, or check if the operation you're trying to execute is permitted at all (e.g. calling `system.setCode` to do a runtime upgrade directly, without voting).
BadProof	The transaction's signature seems invalid.	It's possible that the node you're connected to is following an obsolete fork - trying again after it catches up usually resolves the issue. To check for bigger problems, inspect the last finalized and current best block of the node you're connected to and compare the values to chain stats exposed by other nodes - are they in sync? If not, try connecting to a different node.
Future	Transaction nonce too high, i.e. it's "from the future".	Reduce the nonce to +1 of current nonce. Check current nonce by inspecting the address you're using to send the transaction.
Stale	Transaction nonce too low.	Increase the nonce to +1 of current nonce. Check current nonce by inspecting the address you're using to send the transaction.

Error	Description	Solution
ExhaustsResources	There aren't enough resources left in the current block to submit this transaction.	Try again in the next block.
Payment	Unable to pay for TX fee.	You might not have enough free balance to cover the fee this transaction would incur.
Temporarily banned	The transaction is temporarily banned.	The tx is already in pool. Either try on a different node, or wait to see if the initial transaction goes through.

Error Table

The below table is a reference to the errors that exists in Polkadot. It is generated from the runtime's metadata.

Pallet	Error	Documentation
System (0)		
	InvalidSpecName (0)	The name of specification does not match between the current runtime and the new runtime.
	SpecVersionNeedsToIncrease (1)	The specification version is not allowed to decrease between the current runtime and the new runtime.

Pallet	Error	Documentation
	FailedToExtractRuntimeVersion (2)	Failed to extract the runtime version from the new runtime. Either calling `Core_version` or decoding `RuntimeVersion` failed.
	NonDefaultComposite (3)	Suicide called when the account has non-default composite data.
	NonZeroRefCount (4)	There is a non-zero reference count preventing the account from being purged.
Scheduler (1)		
	FailedToSchedule (0)	Failed to schedule a call
	NotFound (1)	Cannot find the scheduled call.
	TargetBlockNumberInPast (2)	Given target block number is in the past.
	RescheduleNoChange (3)	Reschedule failed because it does not change scheduled time.
Balances (5)		
	VestingBalance (0)	Vesting balance too high to send value
	LiquidityRestrictions (1)	Account liquidity restrictions prevent withdrawal
	Overflow (2)	Got an overflow after adding
	InsufficientBalance (3)	Balance too low to send value
	ExistentialDeposit (4)	Value too low to create account due to existential deposit

Pallet	Error	Documentation
	KeepAlive (5)	Transfer/payment would kill account
	ExistingVestingSchedule (6)	A vesting schedule already exists for this account
	DeadAccount (7)	Beneficiary account must pre-exist
Authorship (6)		
	InvalidUncleParent (0)	The uncle parent not in the chain.
	UnclesAlreadySet (1)	Uncles already set in the block.
	TooManyUncles (2)	Too many uncles.
	GenesisUncle (3)	The uncle is genesis.
	TooHighUncle (4)	The uncle is too high in chain.
	UncleAlreadyIncluded (5)	The uncle is already included.
	OldUncle (6)	The uncle isn't recent enough to be included.
Staking (7)		
	NotController (0)	Not a controller account.
	NotStash (1)	Not a stash account.
	AlreadyBonded (2)	Stash is already bonded.
	AlreadyPaired (3)	Controller is already paired.
	EmptyTargets (4)	Targets cannot be empty.

Pallet	Error	Documentation
	DuplicateIndex (5)	Duplicate index.
	InvalidSlashIndex (6)	Slash record index out of bounds.
	InsufficientValue (7)	Can not bond with value less than minimum balance.
	NoMoreChunks (8)	Can not schedule more unlock chunks.
	NoUnlockChunk (9)	Can not rebond without unlocking chunks.
	FundedTarget (10)	Attempting to target a stash that still has funds.
	InvalidEraToReward (11)	Invalid era to reward.
	InvalidNumberOfNominations (12)	Invalid number of nominations.
	NotSortedAndUnique (13)	Items are not sorted and unique.
	AlreadyClaimed (14)	Rewards for this era have already been claimed for this validator.
	OffchainElectionEarlySubmission (15)	The submitted result is received out of the open window.
	OffchainElectionWeakSubmission (16)	The submitted result is not as good as the one stored on chain.
	SnapshotUnavailable (17)	The snapshot data of the current window is missing.
	OffchainElectionBogusWinnerCount (18)	Incorrect number of winners were presented.

Pallet	Error	Documentation
	OffchainElectionBogusWinner (19)	One of the submitted winners is not an active candidate on chain (index is out of range in snapshot).
	OffchainElectionBogusCompact (20)	Error while building the assignment type from the compact. This can happen if an index is invalid, or if the weights *overflow*.
	OffchainElectionBogusNominator (21)	One of the submitted nominators is not an active nominator on chain.
	OffchainElectionBogusNomination (22)	One of the submitted nominators has an edge to which they have not voted on chain.
	OffchainElectionSlashedNomination (23)	One of the submitted nominators has an edge which is submitted before the last non-zero slash of the target.
	OffchainElectionBogusSelfVote (24)	A self vote must only be originated from a validator to ONLY themselves.
	OffchainElectionBogusEdge (25)	The submitted result has unknown edges that are not among the presented winners.
	OffchainElectionBogusScore (26)	The claimed score does not match with the one computed from the data.
	OffchainElectionBogusElectionSize (27)	The election size is invalid.

Pallet	Error	Documentation
	CallNotAllowed (28)	The call is not allowed at the given time due to restrictions of election period.
	IncorrectHistoryDepth (29)	Incorrect previous history depth input provided.
	IncorrectSlashingSpans (30)	Incorrect number of slashing spans provided.
Session (9)		
	InvalidProof (0)	Invalid ownership proof.
	NoAssociatedValidatorId (1)	No associated validator ID for account.
	DuplicatedKey (2)	Registered duplicate key.
	NoKeys (3)	No keys are associated with this account.
Grandpa (11)		
	PauseFailed (0)	Attempt to signal GRANDPA pause when the authority set isn't live (either paused or already pending pause).
	ResumeFailed (1)	Attempt to signal GRANDPA resume when the authority set isn't paused (either live or already pending resume).
	ChangePending (2)	Attempt to signal GRANDPA change with one already pending.

Pallet	Error	Documentation
	TooSoon (3)	Cannot signal forced change so soon after last.
	InvalidKeyOwnershipProof (4)	A key ownership proof provided as part of an equivocation report is invalid.
	InvalidEquivocationProof (5)	An equivocation proof provided as part of an equivocation report is invalid.
	DuplicateOffenceReport (6)	A given equivocation report is valid but already previously reported.
ImOnline (12)		
	InvalidKey (0)	Non existent public key.
	DuplicatedHeartbeat (1)	Duplicated heartbeat.
Democracy (14)		
	ValueLow (0)	Value too low
	ProposalMissing (1)	Proposal does not exist
	BadIndex (2)	Unknown index
	AlreadyCanceled (3)	Cannot cancel the same proposal twice
	DuplicateProposal (4)	Proposal already made
	ProposalBlacklisted (5)	Proposal still blacklisted
	NotSimpleMajority (6)	Next external proposal not simple majority

Pallet	Error	Documentation
	InvalidHash (7)	Invalid hash
	NoProposal (8)	No external proposal
	AlreadyVetoed (9)	Identity may not veto a proposal twice
	NotDelegated (10)	Not delegated
	DuplicatePreimage (11)	Preimage already noted
	NotImminent (12)	Not imminent
	TooEarly (13)	Too early
	Imminent (14)	Imminent
	PreimageMissing (15)	Preimage not found
	ReferendumInvalid (16)	Vote given for invalid referendum
	PreimageInvalid (17)	Invalid preimage
	NoneWaiting (18)	No proposals waiting
	NotLocked (19)	The target account does not have a lock.
	NotExpired (20)	The lock on the account to be unlocked has not yet expired.
	NotVoter (21)	The given account did not vote on the referendum.
	NoPermission (22)	The actor has no permission to conduct the action.
	AlreadyDelegating (23)	The account is already delegating.

Pallet	Error	Documentation
	Overflow (24)	An unexpected integer overflow occurred.
	Underflow (25)	An unexpected integer underflow occurred.
	InsufficientFunds (26)	Too high a balance was provided that the account cannot afford.
	NotDelegating (27)	The account is not currently delegating.
	VotesExist (28)	The account currently has votes attached to it and the operation cannot succeed until these are removed, either through `unvote` or `reap_vote`.
	InstantNotAllowed (29)	The instant referendum origin is currently disallowed.
	Nonsense (30)	Delegation to oneself makes no sense.
	WrongUpperBound (31)	Invalid upper bound.
	MaxVotesReached (32)	Maximum number of votes reached.
	InvalidWitness (33)	The provided witness data is wrong.
	TooManyProposals (34)	Maximum number of proposals reached.
Council (15)		
	NotMember (0)	Account is not a member

Pallet	Error	Documentation
	DuplicateProposal (1)	Duplicate proposals not allowed
	ProposalMissing (2)	Proposal must exist
	WrongIndex (3)	Mismatched index
	DuplicateVote (4)	Duplicate vote ignored
	AlreadyInitialized (5)	Members are already initialized!
	TooEarly (6)	The close call was made too early, before the end of the voting.
	TooManyProposals (7)	There can only be a maximum of `MaxProposals` active proposals.
	WrongProposalWeight (8)	The given weight bound for the proposal was too low.
	WrongProposalLength (9)	The given length bound for the proposal was too low.
TechnicalCommittee (16)		
	NotMember (0)	Account is not a member
	DuplicateProposal (1)	Duplicate proposals not allowed
	ProposalMissing (2)	Proposal must exist
	WrongIndex (3)	Mismatched index
	DuplicateVote (4)	Duplicate vote ignored
	AlreadyInitialized (5)	Members are already initialized!

Pallet	Error	Documentation
	TooEarly (6)	The close call was made too early, before the end of the voting.
	TooManyProposals (7)	There can only be a maximum of `MaxProposals` active proposals.
	WrongProposalWeight (8)	The given weight bound for the proposal was too low.
	WrongProposalLength (9)	The given length bound for the proposal was too low.
ElectionsPhragmen (17)		
	UnableToVote (0)	Cannot vote when no candidates or members exist.
	NoVotes (1)	Must vote for at least one candidate.
	TooManyVotes (2)	Cannot vote more than candidates.
	MaximumVotesExceeded (3)	Cannot vote more than maximum allowed.
	LowBalance (4)	Cannot vote with stake less than minimum balance.
	UnableToPayBond (5)	Voter can not pay voting bond.
	MustBeVoter (6)	Must be a voter.
	ReportSelf (7)	Cannot report self.

Pallet	Error	Documentation
	DuplicatedCandidate (8)	Duplicated candidate submission.
	MemberSubmit (9)	Member cannot re-submit candidacy.
	RunnerSubmit (10)	Runner cannot re-submit candidacy.
	InsufficientCandidateFunds (11)	Candidate does not have enough funds.
	NotMember (12)	Not a member.
	InvalidCandidateCount (13)	The provided count of number of candidates is incorrect.
	InvalidVoteCount (14)	The provided count of number of votes is incorrect.
	InvalidRenouncing (15)	The renouncing origin presented a wrong `Renouncing` parameter.
	InvalidReplacement (16)	Prediction regarding replacement after member removal is wrong.
Treasury (19)		
	InsufficientProposersBalance (0)	Proposer's balance is too low.
	InvalidIndex (1)	No proposal or bounty at that index.
	ReasonTooBig (2)	The reason given is just too big.
	AlreadyKnown (3)	The tip was already found/started.

Pallet	Error	Documentation
	UnknownTip (4)	The tip hash is unknown.
	NotFinder (5)	The account attempting to retract the tip is not the finder of the tip.
	StillOpen (6)	The tip cannot be claimed/closed because there are not enough tippers yet.
	Premature (7)	The tip cannot be claimed/closed because it's still in the countdown period.
	UnexpectedStatus (8)	The bounty status is unexpected.
	RequireCurator (9)	Require bounty curator.
	InvalidValue (10)	Invalid bounty value.
	InvalidFee (11)	Invalid bounty fee.
	PendingPayout (12)	A bounty payout is pending. To cancel the bounty, you must unassign and slash the curator.
Claims (24)		
	InvalidEthereumSignature (0)	Invalid Ethereum signature.
	SignerHasNoClaim (1)	Ethereum address has no claim.
	SenderHasNoClaim (2)	Account ID sending tx has no claim.
	PotUnderflow (3)	There's not enough in the pot to pay out some unvested amount. Generally implies a logic error.

Pallet	Error	Documentation
	InvalidStatement (4)	A needed statement was not included.
	VestedBalanceExists (5)	The account already has a vested balance.
Vesting (25)		
	NotVesting (0)	The account given is not vesting.
	ExistingVestingSchedule (1)	An existing vesting schedule already exists for this account that cannot be clobbered.
	AmountLow (2)	Amount being transferred is too low to create a vesting schedule.
Identity (28)		
	TooManySubAccounts (0)	Too many subs-accounts.
	NotFound (1)	Account isn't found.
	NotNamed (2)	Account isn't named.
	EmptyIndex (3)	Empty index.
	FeeChanged (4)	Fee is changed.
	NoIdentity (5)	No identity found.
	StickyJudgement (6)	Sticky judgement.
	JudgementGiven (7)	Judgement given.
	InvalidJudgement (8)	Invalid judgement.
	InvalidIndex (9)	The index is invalid.

Pallet	Error	Documentation
	InvalidTarget (10)	The target is invalid.
	TooManyFields (11)	Too many additional fields.
	TooManyRegistrars (12)	Maximum amount of registrars reached. Cannot add any more.
	AlreadyClaimed (13)	Account ID is already named.
	NotSub (14)	Sender is not a sub-account.
	NotOwned (15)	Sub-account isn't owned by sender.
Proxy (29)		
	TooMany (0)	There are too many proxies registered or too many announcements pending.
	NotFound (1)	Proxy registration not found.
	NotProxy (2)	Sender is not a proxy of the account to be proxied.
	Unproxyable (3)	A call which is incompatible with the proxy type's filter was attempted.
	Duplicate (4)	Account is already a proxy.
	NoPermission (5)	Call may not be made by proxy because it may escalate its privileges.
	Unannounced (6)	Announcement, if made at all, was made too recently.
Multisig (30)		

Pallet	Error	Documentation
	MinimumThreshold (0)	Threshold must be 2 or greater.
	AlreadyApproved (1)	Call is already approved by this signatory.
	NoApprovalsNeeded (2)	Call doesn't need any (more) approvals.
	TooFewSignatories (3)	There are too few signatories in the list.
	TooManySignatories (4)	There are too many signatories in the list.
	SignatoriesOutOfOrder (5)	The signatories were provided out of order; they should be ordered.
	SenderInSignatories (6)	The sender was contained in the other signatories; it shouldn't be.
	NotFound (7)	Multisig operation not found when attempting to cancel.
	NotOwner (8)	Only the account that originally created the multisig is able to cancel it.
	NoTimepoint (9)	No timepoint was given, yet the multisig operation is already underway.
	WrongTimepoint (10)	A different timepoint was given to the multisig operation that is underway.
	UnexpectedTimepoint (11)	A timepoint was given, yet no multisig operation is underway.

Pallet	Error	Documentation
	WeightTooLow (12)	The maximum weight information provided was too low.
	AlreadyStored (13)	The data to be stored is already stored.

Bug Bounty

We call on our community and all bug bounty hunters to help identify bugs in Kusama.

If you discover a bug, we appreciate your cooperation in responsibly investigating and reporting it as per instructions on the Web3 Foundation website. Disclosure to any third parties disqualifies bug bounty eligibility.

Eligibility

Generally speaking, any bug that poses a significant vulnerability, either to the soundness of protocols and protocol/implementation compliance to network security, to classical client security as well as security of cryptographic primitives, could be eligible for reward. Please note that it's entirely at our discretion to decide whether a bug is significant enough to be eligible for reward.

Examples include: An attack that could disrupt the entire network and harm the validity to the network would be considered a critical threat. An attack that would cause disruption in service to others would be considered a high threat.

Please note: The submission quality will be a large factor in the level of considered compensation. A high quality submission includes an explanation of how the bug can be reproduced, how it was discovered and elsewise critical details. Please disclose responsibly; disclosure to any third parties disqualifies bug bounty eligibility.

Responsible investigation and reporting Responsible investigation and reporting includes, but isn't limited to, the following:

- Don't violate the privacy of other users, destroy data, etc.
- Don't defraud or harm Kusama network or its users during your research; you should make a good faith effort to not interrupt or degrade our services.
- Don't target the validators' physical security measures, or attempt to use social engineering, spam, distributed denial of service (DDoS) attacks, etc.
- Initially report the bug only to us and not to anyone else.
- Give us a reasonable amount of time to fix the bug before disclosing it to anyone else, and give us adequate written warning before disclosing it to anyone else.
- In general, please investigate and report bugs in a way that makes a reasonable, good faith effort not to be disruptive or harmful to us or our users. Otherwise your actions might be interpreted as an attack rather than an effort to be helpful.

How to report a bug

Please follow the instructions at web3.foundation/security-report/.

Adversarial Cheatsheet

Expect things to break on Kusama. To help you break some things, take a look at the following threat model.

Hacker wants to ...	Security promise that should prevent the hack	Hacking Incentive	Hacking Damage	Hacking value details
Double spend tokens via getting the clients to accept a different chain	Integrity (System-wide)	High	High	If attackers are able to double spend tokens, they are able to get services without paying for them. This gives them a high monetary incentive to execute the attack.
Cause system to mint tokens to his own account	Integrity (System-wide)	Medium	Low - Medium	If an attacker is able to craft transactions that mint tokens to their account, then this provides a high monetary incentive to execute this attack.
Validate malicious blocks to double spend tokens	Availability (System-wide)	High	Medium	If an attacker is able to double spend tokens, they are able to get services without paying for them. This gives them a high monetary incentive to execute the attack.

Hacker wants to ...	Security promise that should prevent the hack	Hacking Incentive	Hacking Damage	Hacking value details
Undermine consensus mechanism to split chain	Integrity (System-wide)	High	High	"If an attacker is able to double spend tokens, they are able to get services without paying for them. This gives them a high monetary incentive to execute the attack. Betting on decrease in value of the cryptocurrency or competitors want to damage the reputation, so that the value of their blockchain increases.
Tamper/manipulate blockchain history to invalidate transactions (e.g. a voting result)	Integrity (System-wide)	Medium	Medium - High	Attacker can rollback undesired transactions by intentionally invalidating the block where transaction has happened. Attacker can force a governance decision (or even an on-chain update) that favors them.
Undermine blockchain or consensus mechanism to damage the ecosystem's reputation	Availability (System-wide)	High	High	Betting on decrease in value of the cryptocurrency or competitors want to damage the reputation, so that the value of their blockchain increases

Hacker wants to ...	Security promise that should prevent the hack	Hacking Incentive	Hacking Damage	Hacking value details
Censorship	Availability (System-wide)	Medium	High	Hackers are able to block undesirable types of transactions (e.g. industry competitor transactions or referendum votes). This could be achieved by colluding with other stakeholders or by otherwise obtaining more voting power.
Deanonymize users	Confidentiality (Node)	Medium	Medium	Parties that want to de-anonymize users can use the information to oppress the opposition (e.g. political activists).
Steal token from node	Integrity (Node)	High	High	Attackers that are able to steal tokens from nodes can claim assets for themselves, which gives them a high monetary incentive to execute the attack.
Steal token from node by leaking credentials	Confidentiality (Node)	High	High	Attackers that are able to steal tokens from nodes can claim assets for themselves, which gives them a high monetary incentive to execute the attack.
Prevent node from accessing the Polkadot network	Availability (Node)	Low	Low - Medium	Run a targeted denial-of-service attack out of revenge, monetary interests (in case of a competing coin exchange, etc.).

Hacker wants to ...	Security promise that should prevent the hack	Hacking Incentive	Hacking Damage	Hacking value details
Defraud other participants	Integrity (Node)	Medium	Low - Medium	Attacker can abuse other participants' misunderstanding of Polkadot's security guarantees to defraud them. Also, if the reward for calling out bad behavior can be set up so that it is higher than the according punishment, a set of self-handled nodes can be set up to generate a source cycle. Other participants are not needed for this attack.
Defraud other participants	Integrity (System-wide)	High	High	An attacker could abuse bugs in Polkadot's economic system to defraud other participants. For example, an attacker could exploit a logic bug to not pay transaction fees.

Adding accounts to an ENS domain

ENS (Ethereum Name Service) is a system of smart contracts on the Ethereum blockchain which allows users to claim domain names like `bruno.eth`. Supporting wallets can then allow senders to input ENS domains instead of long and unwieldy addresses. This prevents phishing, fraud, typos, and adds a layer of usability on top of the regular wallet user experience.

> Note: You will need an ENS name and an Ethereum account with some ether in it to follow along with this guide. To register an ENS name, visit the ENS App or any number of subdomain registrars like Nameth. Note that if you're using an older ENS name, you should make sure you're using the new resolver. Visiting the ENS App will warn you about this if not. You will also need some way to use your Ethereum address - following this guide on a personal computer is recommended. Wallets like Frame and Metamask are safe and will make interacting with the Ethereum blockchain through your browser very easy.

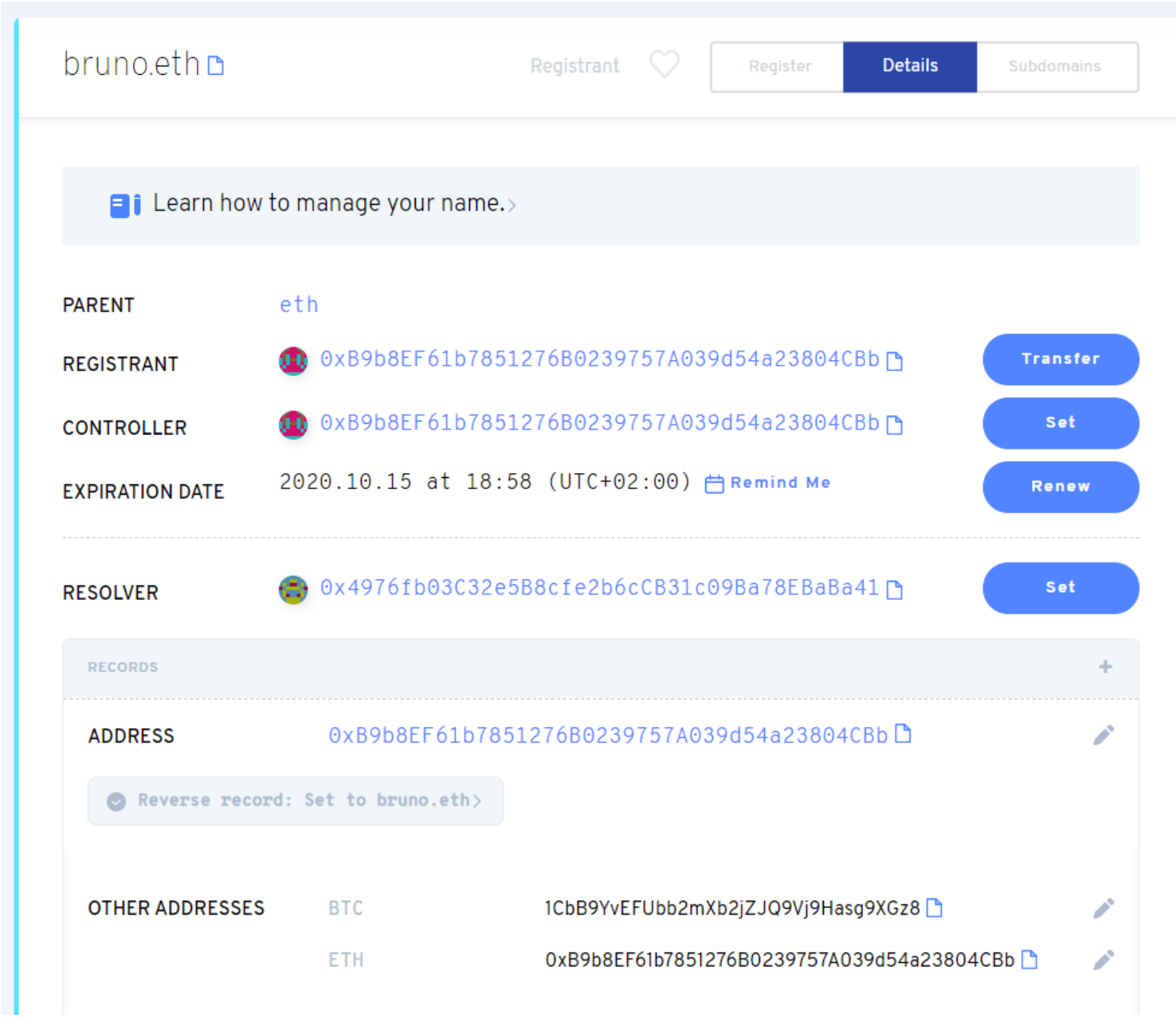
Despite living on the Ethereum blockchain, the ENS system has multi-chain support. In this guide you'll go through the process of adding a KSM and DOT address to ENS. We cover both KSM and DOT to show two different approaches.

> Note: DOT can currently only be added using the Resolver method. KSM can be added through both methods described below.

This guide is also available in video format on Youtube.

Adding via the UI

The ENS App allows an ENS domain owner to inspect all records bound to the domain, and to add new ones.



In the example above, the domain `bruno.eth` has an Ethereum and a Bitcoin address attached. Let's attach a KSM account. First, click the `[+]` icon in the Records tab.

Then, pick "Other Addresses", "KSM", and input the Kusama address:

After clicking Save, your Ethereum wallet will ask you to confirm a transaction. Once processed, the record will show up on the domain's page:

The same process applies to adding your DOT address.

Once the transaction is confirmed, your address will be bound to your ENS domain.

Wallet Support

There is no wallet support for ENS names for either KSM or DOT at this time, but the crypto accounting and portfolio application Rotki does support KSM ENS resolution.

Relevant links

- ENS docs
- ENS Multi-chain announcement
- Address encoder
- Namehash calculator
- Address to pubkey converter

Using Ledger Devices

> Note: Because of required WebUSB support, Ledger wallets currently only work on Chromium-based browsers like Brave and Chrome.

Kusama has a Ledger application that is compatible with the Ledger Nano S and Ledger Nano X devices. The Ledger devices are hardware wallets that keep your private key secured on a physical device that does not get directly exposed to your computer or the internet.

The Kusama application allows you to manage Kusama's native asset, the KSM token. It supports most of the available transaction types of the network (a notable exception is the "Batch" transaction from the Utility pallet).

If you have trouble using Ledger or following the directions below, you can try searching for your issue on the Polkadot Knowledge Base.

> Please check out our intro to Ledger video on Youtube for more information.

Requirements

Here is a list of what you will need before starting:

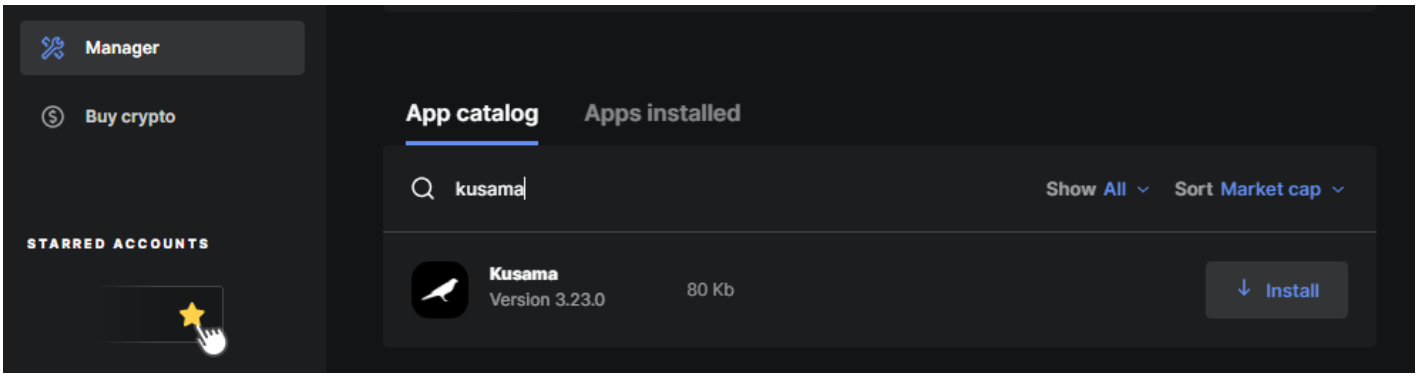
- A Ledger Nano S or a Ledger Nano X.
- The latest firmware installed.
- Ledger Live is installed and at version 2.19 or newer (see settings -> about to find out if you're up to date).
- A web browser is installed that you can use to access Polkadot-JS Apps UI.

Installing the Ledger Application

Using Ledger Live

- Open the "Manager" tab in Ledger Live.
- Connect and unlock your Ledger device.
- If asked, allow the manager on your device by pressing both buttons on the YES screen.

- Find Kusama in the app catalog and install it.



Please proceed to the usage instructions below.

Using the Developer Release

> NOTE: These instructions are for development installation only. It is recommended to install the application from Ledger Live unless you *know exactly what you're doing*.

Instructions for downloading the prerelease binary from the GitHub releases is written on the README for the Kusama Ledger application GitHub repository.

On the releases page you can download the shell script `install_app.sh` and then make it executable in your shell by typing the command `chmod +x install_app.sh`.

Using `install_app.sh` help command will show you the available options:

```
$ ./install_app.sh --help
Zondax Installer [Kusama-1.2011.1] [Warning: use only for test/demo apps]
  load    - Load Kusama app
  delete  - Delete Kusama app
  version - Show Kusama app version
```

Next, you must make sure your Ledger device is plugged in and unlocked and you're using the latest firmware. If everything is prepared, then type `./install_app.sh load` and accept the prompts on your Ledger device to install the application.

First it will prompt you to allow an unsafe manager - confirm this by switching the screen to the allow screen and pressing the corresponding buttons.

After some processing time, the screen of your device will update to say "Install app Kusama". Navigate all the way to the right, verify the Identifier hash matches the one that is printed in your terminal. Click both buttons on "Perform Installation" to install the application. It will ask again for your Pin code and you should enter it in the device.

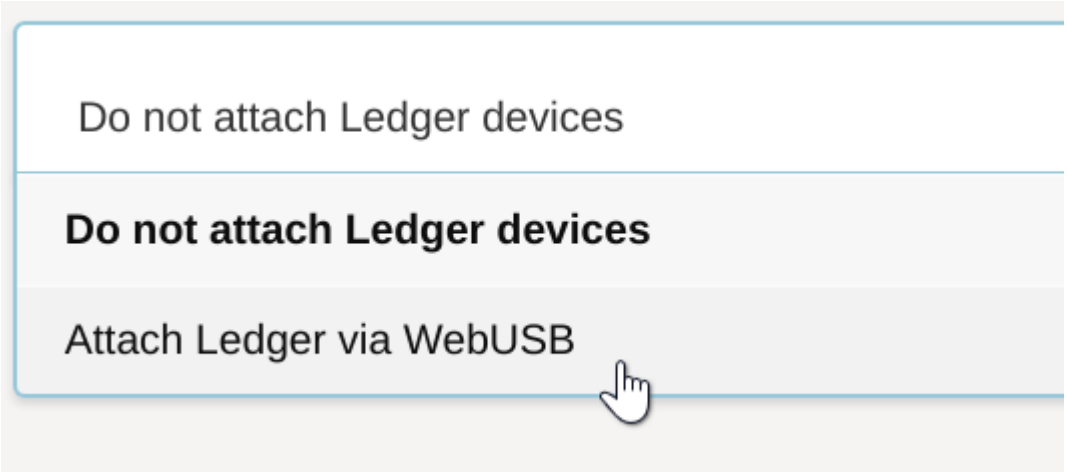
At the end of the process you should have the newly installed Kusama application on the device.

Using on Polkadot-JS Apps UI

Loading Your Account

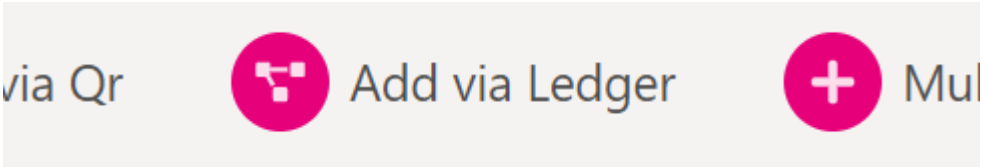
Polkadot-JS Apps UI already has an integration with the Ledger application so that your device will work with the browser interface after installation. The functionality is currently gated behind a feature setting that you will need to turn on.

In order to turn on the interoperability with the Kusama Ledger application, go to the "Settings" tab in Polkadot-JS Apps UI. Find the option for attaching Ledger devices and switch the option from the default "Do not attach Ledger devices" to "Attach Ledger via WebUSB".

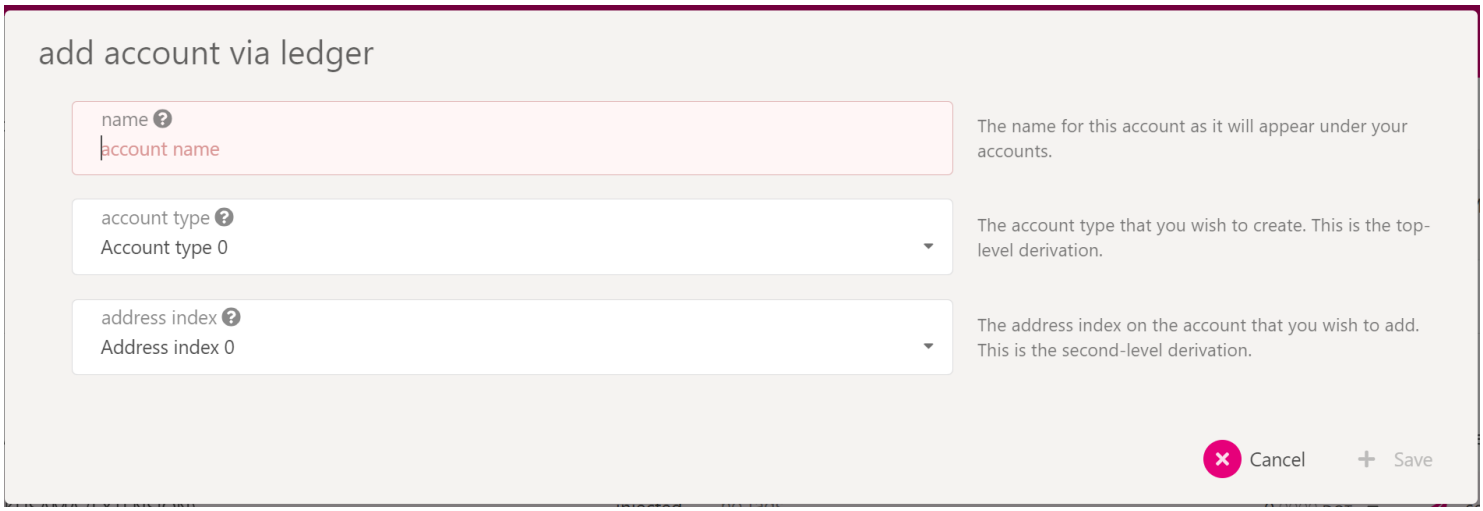


Click "Save" to keep your settings.

Now when you go to the "Accounts" tab you will see a new button that says "Add Ledger". Ensure that your Ledger device is unlocked and you have navigated into the Kusama application, then click this button.

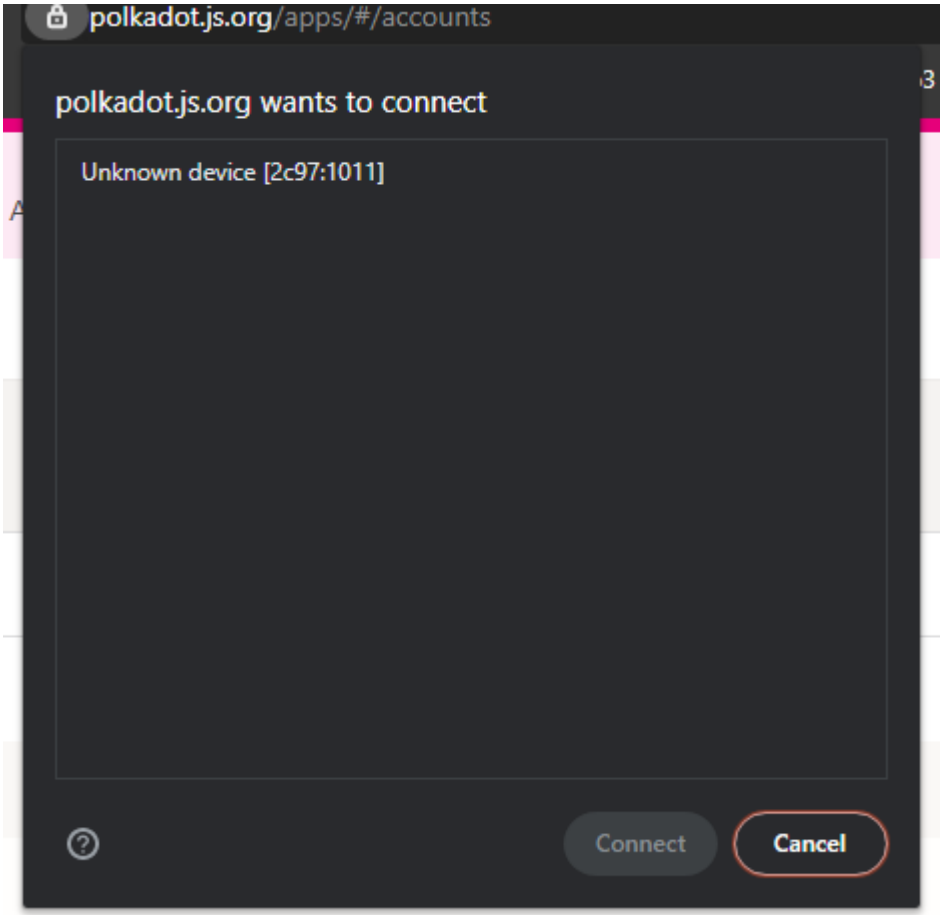


A popup will appear asking you to select an account and derivation path.

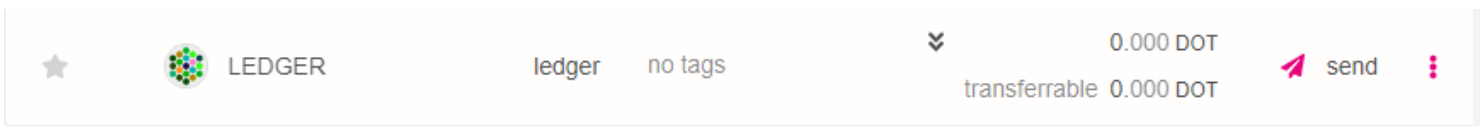


The first option lets you select an account. You can have multiple accounts on a single Ledger device. The second dropdown lets you pick a derivation path - think of it like a formula from which child accounts are generated. If in doubt, pick the first option for both.

Once you confirm your selection, depending on your browser and its security settings, you might need to confirm the USB connection through a popup like the one below when adding the Ledger device for the first time:



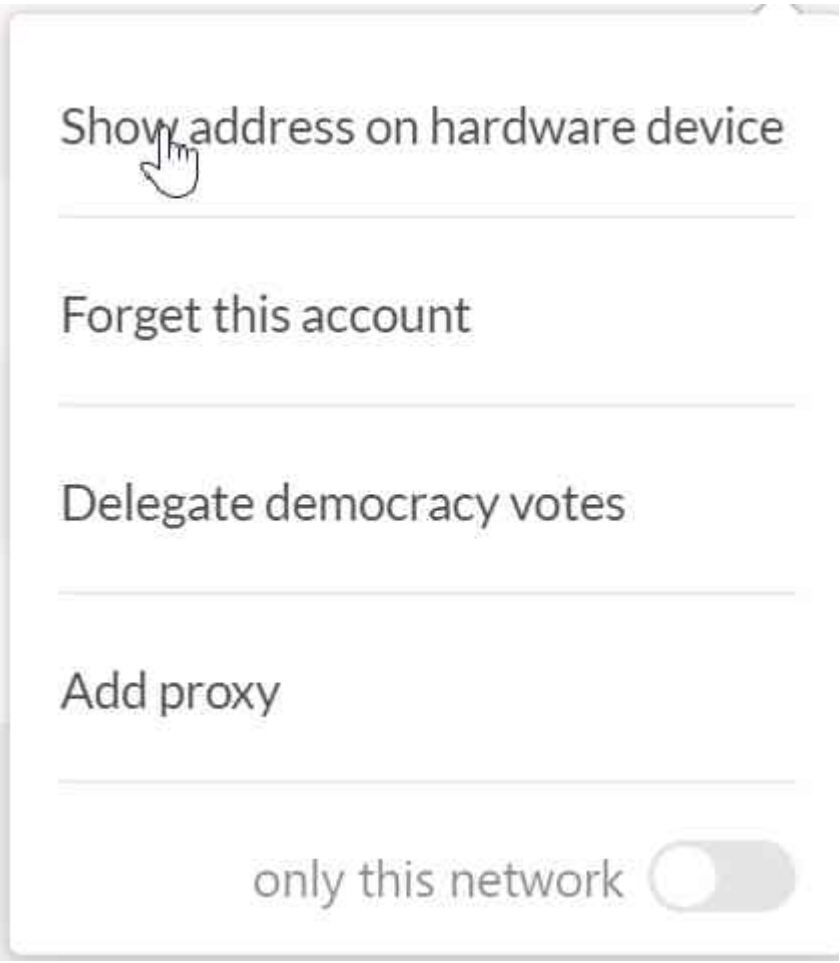
You should now be able to scroll down and find a new account on the page with the type "ledger".



You can now use this account to interact with Kusama on Polkadot-JS Apps UI and it will prompt your ledger for confirmation when you initiate a transaction.

Confirming the Address on your Device

On the "Accounts" tab, find your Ledger-connected account. Click on the three vertical dots at the end of the row. This will open a new menu, here you can click the "Show address on hardware device" option to display the address on your device.



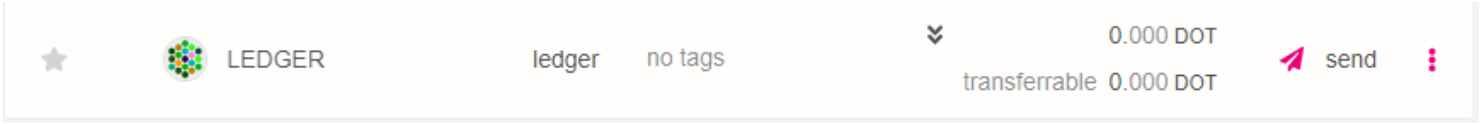
Here you can scroll through and make sure the address matches to what is displayed on Polkadot-JS Apps UI.

Checking the Balance of Your Account

There are a few methods to check the balance of your account. You can use Polkadot-JS Apps or you can use a block explorer like Polkascan or Subscan.

Using Polkadot-JS Apps

Once you have your account loaded on the "Accounts" tab it should show a row with your Ledger account. At the far right of the row is located your account's DOT balance. If you expand the balance arrow, it will show details of your balance such as locks or reserved amounts.



Sending a Transfer

If you would like to send a transfer from your account housed on the Ledger device, the easiest method is to use Polkadot-JS Apps UI.

- Click on the "Transfer" button in "Accounts" dropdown in the top navigation menu.
- In the top input, select "Ledger" as your sending account.
- In the second input, select the account that you want to transfer funds to.
- In the third input, enter the amount of KSM you want to transfer.
- Click the "Make Transfer" button.
- Confirm the transaction on your device.
- A green success notification will be displayed when the transaction is included in a block.

> Note the "Transfer with Keep-Alive Checks" toggle. While this toggle is in the *On* state, your account will be unable to make transactions which would get its balance below the existential deposit. This prevents reaping of accounts with low balances. If you toggle this to *Off*, you will be able to go below existential deposit balance, causing your account to be deleted and any dust amount of KSM to be burned. If you encounter KeepAlive errors when making transactions, this might be the reason.

A detailed guide on doing transfers is available here.

Receiving a Transfer

In order to receive a transfer on the accounts stored on your Ledger device, you will need to provide the sender (i.e. the payer) with your address.

The easiest way to get your address is to click on the account name which will open a sidebar. Your address will be shown in this sidebar, along with some other information. Another method is just clicking on your account's avatar icon - this immediately copies your address to the clipboard.

> Warning: before giving anyone your address, make sure it matches what's really on the Ledger by confirming the address on your device. Some malware will intercept clicks and clipboard requests and can change your copied value in-flight, so being extra vigilant around copy-paste operations makes sense.

Staking

Since Ledger does not support batch transactions, you must do two separate transactions when you want to stake using an account stored on a Ledger device.

- Go to the "Staking" tab found under the "Network" dropdown in the top navigation menu.
- Click the "Account Actions" pane in the inner navigation.
- Click "+ Stash" instead of "+ Nominator" or "+ Validator" (selecting the latter two will not work).
- Input the amount of tokens to bond and confirm the transaction.
- Confirm the transaction on the Ledger devivce.
- When the transaction is included you will see the newly bonded account in the "Account Actions" page.
- Select "Start Nominating" or "Start Validating" to start nominating or validating.
- Confirm the transaction on Apps and on the Ledger device.

Support

If you need support please send an email to support@kusama.network or visit our Support page.

Kusama Timeline

Kusama network started as a Proof-of-Authority network and was transitioned to Proof-of-Stake on 28 October 2019 at approximately 18:43 Zurich time (CET). The first successful validator set rotation took place at 22:45 CET.

Currently, Kusama is a healthy Proof-of-Stake network with over 500 validators and over two million blocks produced. If you are curious about the history of the Kusama network, you will find more information in the sections below.

Rollout plan

The rollout of full functionality of Kusama was staggered to allow for a safe transition. The first PoS phase began with 20 validators. Of the 20, the Web3 Foundation ran nine and Parity Technologies ran six. Five were ran by highly staked community members as voted in by the Phragmén election.

When the initial transition was successful, additional validator spots were opened 10 at a time in order to allow for more validators to enter the active set.

When the first transition to PoS took place, the full functionality of Kusama was not fully available. Notably, the Sudo key still existed and was used to initiate further upgrades. Balance transfers were still disabled for a short while.

Kusama now has its full functionality enabled.

Kusama's First Adventure

Source

On 4th January 2020, the Polkadot mainnet runtime, which at that time still wasn't live, was uploaded to the Kusama chain during a runtime upgrade. The mishap was due to a recent split of the Kusama logic from the Polkadot logic and that runtime was not properly named. This led to a halt of block production on the Kusama chain and bricked the chain entirely.

The solution to the issue involved a rollback of the chain history before the problematic runtime upgrade took place. However, due to intricacies of the block production mechanism, it was also necessary to encapsulate the validators of the chain into a time bubble to trick them into believing that they were producing blocks in the past. Furthermore, in order for the chain to catch up to the

present moment it was necessary to make time flow in the bubble at a speed of six times greater than the speed of time in the real world. Therefore, the session of Kusama which would normally last one hour would last only 10 minutes until the validators caught up to the present moment.

The above plan was executed successfully on the 7th of January. Due to the time warp, the number of missed blocks in the sessions directly following block #516558 was significantly higher. This is partly what contributes to the much higher ratio of missed blocks on Kusama versus Polkadot today.

Kusama Community

The following is a list of official chats, forums, and social channels for the Kusama community. Keep in mind that no admin or moderator will ever DM you for any reason whatsoever without prior public contact and anyone doing so is likely trying to scam you.

General

- Kusama Discussion and Governance on Polkassembly
- Kusama forum
- Polkadot GitHub - Parity maintained repository that houses the Rust implementation of the Polkadot and Kusama Host.
- Polkadot Meetup Hub - Information on hosting meetups, applying for funding, and materials for running it.
- Support Knowledgebase and Support Email
- Polkadot's Latest Research - also applies to Kusama.

Events

The Web3 Foundation hosts many events online and in-person. To check out our current and past events, please refer to our public Notion page here.

Matrix Chats

We primarily use Matrix across the organization and to communicate with community members. The application we use most often to interact with the Matrix protocol is the Element messenger.

- Kusama Watercooler Chat - General room for talk about Kusama.
- Kusama Validator Lounge - Room for validators learning about setting up a node.
- Kusama Direction - Governance, and a place to discuss the future of Kusama.
- Polkadot Digest - News about what is happening in the Polkadot ecosystem, published every weekday except holidays, includes Kusama.
- To join the Chinese Validator chat, message Anson
- Kusama Discord

Technical

- Substrate Technical - More advanced room for technical questions on building with Substrate.
- Smart Contracts & Parity Ink! - A room to discuss developing Substrate smart contracts using Parity Ink!

Social

- Twitter: @kusamanetwork
- Reddit: r/kusama
- Kusama YouTube Channel

Blogs and tutorials

- Web3 Medium Blog
- Polkadot Blog
- Gavin Wood's Medium Blog
- Dotleap.com Tutorials

Newsletters

- Subscribe to the Polkadot newsletter - official, infrequent, includes Kusama.
- Dot Leap Newsletter - less official, weekly.
- NFT Review - Covering the evolution of the NFT ecosystem on Kusama

Kusama Parameters

Many of these parameter values can be updated via on-chain governance. If you require absolute certainty of these parameter values, it is recommended you directly check the constants by looking at the chain state and/or storage.

Periods of common actions and attributes

NOTE: Kusama generally runs 4x as fast as Polkadot, except in the time slot duration itself. See Polkadot Parameters for more details on how Kusama's parameters differ from Polkadot's.

- Slot: 6 seconds *(generally one block per slot, although see note below)
- Epoch: 1 hour (600 slots x 6 seconds)
- Session: 1 hour (6 sessions per Era)
- Era: 6 hours (3600 slots x 6 seconds)

Kusama	Time	Slots*
Slot	6 seconds	1
Epoch	1 hour	600
Session	1 hour	600
Era	6 hours	3_600

**A maximum of one block per slot can be in a canonical chain. Occasionally, a slot will be without a block in the chain. Thus, the times given are estimates. See Consensus for more details.*

Governance

Democracy	Time	Slots	Description
Voting period	7 days	100_800	How long the public can vote on a referendum.

Democracy	Time	Slots	Description
Launch period	7 days	100_800	How long the public can select which proposal to hold a referendum on, i.e., every week, the highest-weighted proposal will be selected to have a referendum.
Enactment period	8 days	115_200	Time it takes for a successful referendum to be implemented on the network.

Council	Time	Slots	Description
Term duration	1 day	14_400	The length of a council member's term until the next election round.
Voting period	1 day	14_400	The council's voting period for motions.

The Kusama Council consists of up to 19 members and up to 19 runners up.

Technical committee	Time	Slots	Description
Cool-off period	7 days	604_800	The time a veto from the technical committee lasts before the proposal can be submitted again.
Emergency voting period	3 hours	1_800	The voting period after the technical committee expedites voting.

Staking, Validating, and Nominating

Kusama	Time	Slots	Description
Term duration	6 hours	3_600	The time for which a validator is in the set after being elected. Note, this duration can be shortened in the case the a validator misbehaves.
Nomination period	6 hours	3_600	How often a new validator set is elected.

Kusama	Time	Slots	Description
Bonding duration	7 days	604_800	How long until your funds will be transferrable after unbonding. Note that the bonding duration is defined in eras, not directly by slots.
Slash defer duration	7 days	604_800	Prevents overslashing and validators "escaping" and getting their nominators slashed with no repercussions to themselves. Note that the bonding duration is defined in eras, not directly by slots.

Treasury

Treasury	Time	Slots	Description
Periods between spends	6 days	86_400	When the treasury can spend again after spending previously.

Burn percentage is currently `0.20%`, though instead of being burned this amount is temporarily redirected into the Society's treasury to fund growth.

Precision

KSM have 12 decimals of precision. In other words, 1e12 (1_000_000_000_000, or one trillion) Plancks make up a single KSM.

Kusama Code of Conduct

While Kusama has been its own network and has had its own ecosystem for a while now, it is still one big experiment and we need your participation for it to continue being a great success.

Community

We want to foster a sense of collaboration amongst ourselves and the open source community. Kusma participants exist to encourage the open exchange of ideas and expression and require an environment that recognizes the inherent worth of every person and group. We are here to collaborate, discuss, and even disagree. The key is doing this is a manner that comes from a place of respect and professionalism. Participants in the Kusama network may consist of participants in an online forum, on-chain governance participants, parachain deployment teams, validators, enthusiasts, and ecosystem tool builders. We expect for there to be participation from all backgrounds but like to set some social boundaries on how you may be treated and treat others.

Kusama community members come from across the globe and are not bordered by race, gender, or background. Community participants have read through the requisite resources and obtained sufficient knowledge about Kusama and all related content. This knowledge equips the community with the requisite information needed in the dispense of their duties as a participant.

Bugs

Please understand that this network is, despite its success, an experiment with potential flaws, so it's appreciated that community members help report any sort of exploits directly to the team before sharing publicly. Please see the bug bounty program.

Examples of Unacceptable Behavior

Please note: these are just a few examples and you can always consult a team member if you have any questions.

- Angry aggressive comments towards individuals or other projects on any medium of communication.
- Knowingly distributing false information about Kusama or other projects.
- Harassing other individuals or projects.

That said, please note that Kusama is an edgy and meme-friendly network and community member actions shouldn't be taken too seriously - try to assume jest before malice.

What To Do If You Witness or Are Subject To Unacceptable Behavior

If you are being harassed, notice that someone else is being harassed, or have any other concerns relating to harassment, please contact the administrator of the channel you’re in.

This Code of Conduct may be revised at any time. We are always willing to revise this document based on feedback from the Kusama participants and/or the Polkadot community.